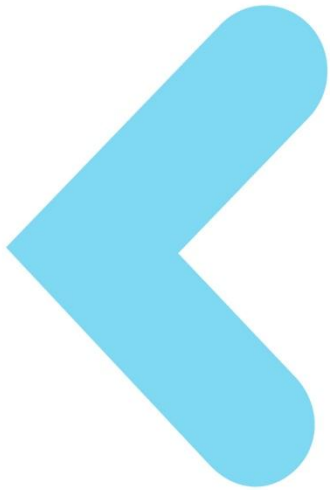
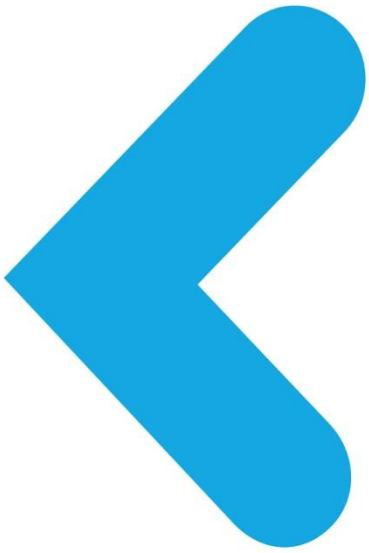
TRUSTONIC



‹t-base Security Architecture

SECURITY ARCHITECTURE



PREFACE

VERSION HISTORY

Version	Date	Modification
1.0	10 Jan 2013	First Issued version
2.0	08 Oct 2013	Added clarifications and minor corrections

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 SCOPE OF THIS DOCUMENT

This document covers the security aspects of the <t-base Trusted Execution Environment in conjunction with a System on Chip incorporating the ARM TrustZone technology. The integration into the secure boot process is described as well as the key infrastructure used in the <t-base architecture. The document is dedicated to technical experts who have to deal with the following topics:

- Integrate <t-base into an existing infrastructure
- Set up a secure infrastructure around <t-base
- Understand the security aspects of <t-base

1.2 GLOSSARY AND ABBREVIATIONS

Activation	Before a Service Provider can install and run TAs, <t-base needs to be activated on the device by the <t-directory Service Enabler. The Root Registration Activation is equivalent to Activation. Activation changes the life cycle state of <t-base to a fully functional state. Note: before activation <t-base may be available to support certain device security features, but will not be able to support Service Provider's trusted services.
Container	Containers are hierarchical data structures holding content management relevant data and are associated to the various roles in the <t-base ecosystem. <t-base containers are the Root Container, Service Provider Containers and TA Containers.
Content Management TA (CMTA)	The Content Management TA is a <t-base System Trusted Application and is responsible for content management operations in cooperation with the NWd Provisioning Agent.
CMP	Content Management Protocol. The interface between the CMTA and the NWd and/or backend.
CSN	Chip Serial Number. The CSN is chip manufacture-wide unique and is used to identified a chip from a chip manufacture. CSN is part of the SUID.
KID	Key Identifier. The KID is used in the Device Bindung process to identify the public key PuK.Kph.Request to be used in the

	signature verification of the command GenerateAuthToken. The signature for GenerateAuthToken is created by the KPH using PrK.Kph.Request.
KPH	Key Provisioning Host – Device used during device binding to issue the K.SoC.Auth.
NWd	Normal-World (non-secure world). The software system running on an ARM TrustZone enabled device in non-secure mode.
PID	Personalization Data Identifier. The PID is an identifier (name) of personalization data loaded into a TA. Several Personalization data blocks with unique PIDs may be loaded into a TA.
Provisioning Agent (PA)	The Provisioning Agent is the NWd counterpart of the SWd Content Management TA and cooperates with that System TA for Content Management operations. The Provisioning Agent communicates with <t-directory and the Service Provider backend systems. Furthermore the Provisioning Agent is responsible to store Content Management Secure Objects in NWd memory space.
Rich OS	The platform OS like Symbian, Android, iOS, … running in the NWd.
Root Container	The Root Container is associated with <t-directory and Root role and is available after <t-base Root Registration. It organizes the Service Provider Containers of the Service Providers registered at the device.
Secure Object, SO	Secure Objects are a mean for the persistent storage of Authentication Token, Root, Service Provider, TA Container, TA Personalization data as well as TA specific confidential data. A Secure Object is an encrypted and integrity protected data package of that data. A Secure Object is transferred from SWd to Nwd to be persisted in NWd storage space.
Service Provider Container	A Service Provider Container is associated to a specific Service Provider holding content management relevant Service Provider data. A Service Provider Container is available after the registration of the Service Provider to the device. The Service Provider Container organizes the TA Containers of its TAs.

SiP	Silicon Provider
SiPID	Silicon Provider ID. The SiPID is worldwide unique and is used to identified a registered Silicon Provider. SiPID is part of the SUID
SPID	Service Provider ID. The SPID is used to identify a registered Service Provider in <t-base. SPID is part of the Service Provider Container.
SUID	SoC Unique Identifier. The SUID is an identifier of the SoC and is unique within all SoCs. As the SoC is built in a device (handset) the SUID is used to identify the device in the <t-base ecosystem.
SWd	Secure World, the software running in secure mode in the ARM TrustZone. <t-base TEE is running in the SWd.
TEE	Trusted Execution Environment. <t-base is an implementation of a TEE. The term TEE is defined in the OMTP and GlobalPlatform specifications.
Trusted Application - TA	A Trusted Application is an application of <t-base running in the Secure-World. A Trusted Application provides a service to its clients running in the Normal-World.
TA Container	A TA Container is a container holding content management relevant data of a Trusted Application (e.g. UUID and the Service Provider's key to encrypt TA code).
TZ	ARM TrustZone
TZPC	Trust Zone Protection Controller. The TZPC controls the configuration of memory regions and peripherals to be secure or non-secure.
UUID	Universal Unique identifier. The UUID is an identifier standard used in software construction, standardized by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE). The UUID in the <t-base ecosystem is used to uniquely identify TAs.

2 <T-BASE PRODUCT AND SECURITY OVERVIEW

2.1 <T-BASE PRODUCT OVERVIEW

<t-base is a portable and open Trusted Execution Environment (TEE) aiming at executing Trusted Applications on a device. It includes also built-in features like cryptography or secure objects. It is a versatile environment that can be integrated on different System on Chip (SoC) supporting the ARM TrustZone technology.

<t-base uses ARM TrustZone to separate the platform into two distinct areas, the Normal-World (NWd) with a conventional rich operation system and rich applications and the Secure-World (SWd).

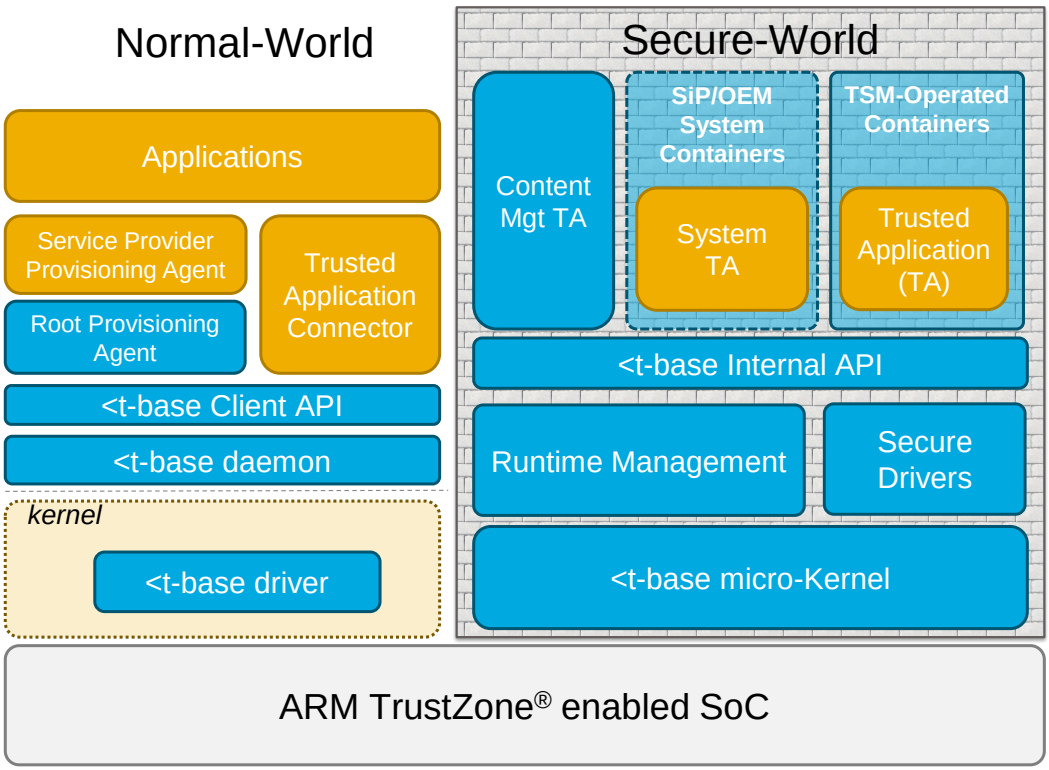


Figure 1: <t-base Architecture Overview.

The Secure-World contains essentially the <t-base core operating system and the Trusted Applications. It provides security functionality to the Normal-World with an on-device client-server architecture. The Normal-World contains mainly software which is not security sensitive (the sensitive code should be migrated to the Secure-World) and it calls the Secure-World to get security functionality via a communication mechanism and several APIs provided by <t-base. The caller in the Normal-World is usually an application, also called a client.

The Trusted Applications in the Secure-World are installed in Containers. A Container is a security domain which can host several Trusted Applications controlled by a third party. There are two kinds of Containers:

- The TSM-Operated Containers, which are created at runtime under the control of Trustonic. Trusted Applications of a TSM-Operated Container can be administrated Over-The-Air via a Trusted Service Manager (TSM).
- The System Container, which is pre-installed at the time of manufacture along with some Trusted Applications. Trusted Applications in the system container cannot be downloaded or updated Over-The-Air via a TSM.

The root Provisioning Agent is a Normal-World component which communicates between the Device and TRUSTONIC's backend system to create TSM-Operated Containers within <t-base at runtime.

Note that in order to enable <t-base and the TSM-Operated Containers on a Device, the OEM must install Trustonic's Key Provisioning Host (KPH) at its manufacturing line. The KPH is a tool which injects a key on the device and which stores a copy in the Trustonic backend system.

<t-base provides APIs in the Normal-World and the Secure-World. In the Normal-World, the <t-base Client API is the API to communicate from the Normal-World to the Secure-World. This API enables to establish a communication channel with the Secure-World and send commands to the Trusted Applications. It enables also to exchange some memory buffers between the Normal-World and the Trusted Applications.

In the Secure-World, <t-base provides the <t-base Internal API which is the interface to be used for the development of Trusted Applications.

Furthermore, the <t-base architecture supports Secure Drivers to interact with any secure peripherals.

2.2 ROLES IN THE <T-BASE ECOSYSTEM

2.2.1 Silicon Provider (SiP)

The Silicon Provider (SiP) develops and maintains the System on Chip (SoC). The SiP provides the SoC to the OEM in order to integrate the SoC into the device. The SiP pre-integrates <t-base on the SoC and with the SoC software to ease the final integration by the OEM. In particular the SiP may develop a reference version of the Secure Boot process (see chapter 4) which can then be customized by the OEM.

2.2.2 TEE Vendor - Trustonic

Trustonic is a TEE Vendor. It is responsible for designing, developing and maintaining <t-base and its related components such as the application programming interfaces or the secure kernel. Trustonic provides <t-base to the SiP for its integration on the SoC.

2.2.3 OEM

The OEM selects the most appropriate SoC for the development and production of the respective device e.g. net books, smart phones, mobile phones. The OEM is authorized by the SiP to extend and modify in particular boot stage BS1 and boot stage BS2 of the secure boot process (see chapter 4.2). During this integration of the device, the OEM must sign the <t-base image and generate the Device Binding Key.

The OEM is also responsible for integration of the Rich OS.

2.2.4 Device

The Device is manufactured by the OEM and belongs to the device owner which is the end-user. The Device incorporates the relevant components to run <t-base and the Trusted Applications. It incorporates a rich operating system such as Android, Linux, Windows and others.

2.2.5 Root - <t-directory

<t-directory is a Service Enabler administrated by Trustonic. It runs and manages the infrastructure required to create Containers in <t-base and to let Service Providers manage their Trusted Applications into their Containers.

2.2.6 Service Provider

The Service Provider provides services for the device such as video on demand, mobile banking etc... The service provided by the Service provider typically consists of a Normal-World application and a corresponding Trusted Application.

The Service Provider entity is responsible for the management of its Trusted Applications. The Service Provider contacts <t-directory to get a Container in <t-base and can then manage its Trusted Applications into its Container.

Note a Service Provider company sometime delegates the management of its Trusted Applications to Trusted Service Manager (TSM) companies which are then playing the role of the Service Provider.

2.3 SECURITY ASPECTS

Within the scope of the present document, the security objective is the protection of assets on a Device. Assets are entities with inherent value. Within the context of <t-base, assets reside and are executed in the Secure-World. Examples of assets are:

- Trusted Applications code or data
- Cryptographic key material
- Certificates
- Etc.

Dedicated countermeasures and mechanisms are implemented within the <t-base system to reduce risk and to eliminate threats to assets. The following mechanisms are taken into account:

- Protection of Secure World data
- Protection against software attacks
- Confidentiality i.e. prevention of unauthorized disclosure of information
- Authenticity
- Integrity i.e. prevention of unauthorized modification of information
- Isolation of Trusted Applications
- System availability i.e. protection against internal Denial Of Service attacks

To provide a broad and robust system security, other relevant aspects have to be considered and defined during the lifecycle of a Device. Those include but are not limited to:

- Key generation mechanisms such as generation of certified key material in a high security and trusted environment
- Physical/logical key exchange procedures between entities within the <t-base ecosystem

Please note that these aspects are out of scope for the present document. Further information can be provided upon request.

3 ARM TRUSTZONE ENABLED SOC

The security of the <t-base TEE is based on the security mechanisms of ARM TrustZone enabled CPU cores in combination with additional peripherals. TrustZone enabled CPU cores are always part of a System on Chip (SoC) which incorporates peripherals such as DRAM controllers, flash controllers, timers, MPUs, keypad and display controllers among others. TrustZone is designed to provide hardware-enforced logical separation between security components and the rest of the SoC infrastructure. With the TrustZone technology, the CPU and its peripherals can be configured to be executed in secure or non-secure mode and can be switched from non-secure mode to secure mode and vice versa.

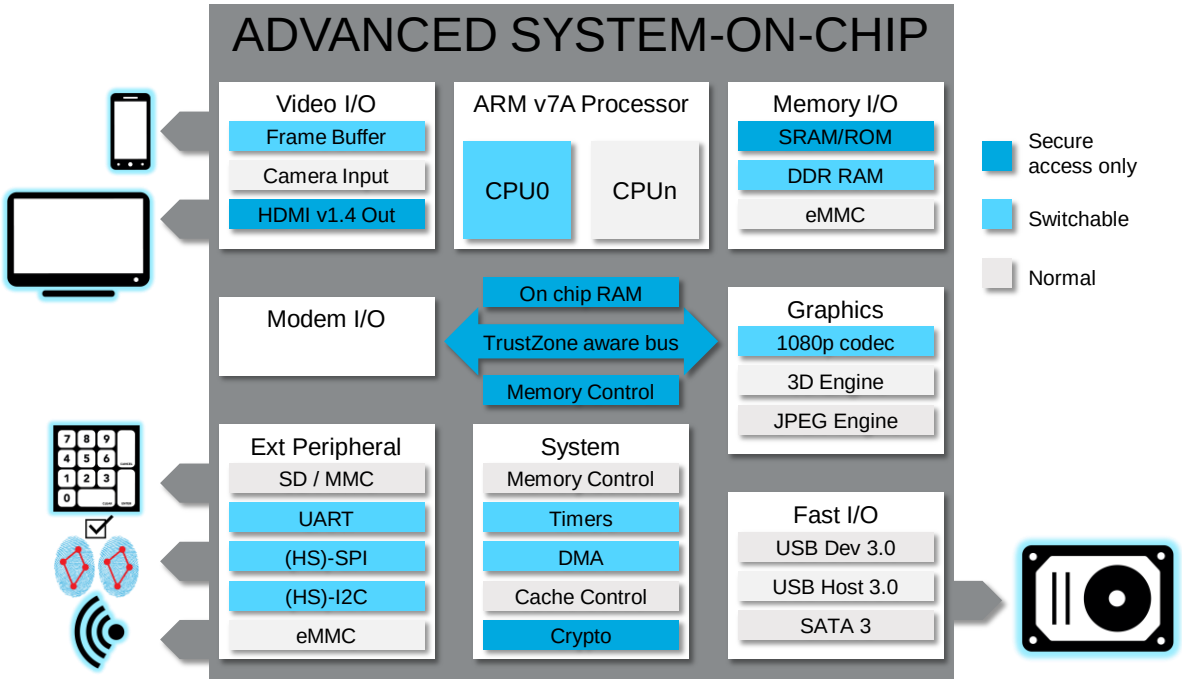


Figure 2: System on Chip with TrustZone.

To understand the security concepts of the <t-base it is essential to understand the basic structure of a System on Chip (SoC) and the TrustZone related concepts.

The reader should refer to the ARM website to get more details about the TrustZone technology. In particular the document "*ARM Security Technology – Building a Secure System using TrustZone Technology*" provides an overview of the ARM TrustZone technology.

3.1 TRUSTZONE PROTECTION CONTROLLER (TZPC)

One of the key peripherals for the security is the ARM TrustZone Protection Controller (TZPC), which splits the software running on the SoC into two execution domains - NWd and SWd.

The TZPC controls the configuration of memory regions and peripherals to be secure or non-secure. It provides a software accessible interface that allows the configuration of the SoC at runtime which is usually done once in an early boot stage in the boot process (see chapter *4.*) of the SoC.

As the TZPC is a peripheral added to the ARM core it is up to the SoC designer to decide which type of controller to integrate. The SoC designer has to take care that the chosen controller provides the required isolation mechanisms.

3.2 SECURE BOOT ROM

The secure boot ROM is embedded as root of trust in the SoC. It contains executable code for Boot Stage 0 (BS0) (see chapter *4.2.1*) and additional information as public keys or hash values for verification of the following boot stages.

3.3 SECURE RAM

Secure RAM can only be accessed by the Secure World (SWd) or by secure peripherals.

Secure peripherals, e.g. a Secure Display or a Secure Keypad, are controlled by <t-base for exclusive temporary access by SWd. In this mode those peripherals may not be accessed by the NWd. When controlled by the SWd those peripherals may have access to Secure RAM.

By configuring the TZPC during the boot process, the available RAM is partitioned into secure and non secure memory. As secure RAM is crucial for the security of the platform the configuration stays unchanged during the power-cycle of the device. Two physical types of RAM can be distinguished:

- On-SoC RAM is located on the same silicon die as the ARM Core or which is on a different silicon die, but inside the same chip package.
- Off-SoC RAM is memory located outside the SoC, e.g. DRAM which is connected to the SoC and driven by a DRAM controller on the SoC. Depending on the characteristics of the DRAM controller, such Off-SoC memory may be configured as secure memory during the boot process.

4 SECURE BOOT PROCESS

The description of the boot process in this chapter serves as an **example**. The implementation of a boot process is OEM specific and may be different for each OEM.

The secure boot process is key for the security of <t-base. The secure boot process must verify the integrity and authenticity of <t-base. The secure boot starts with a secure anchor, which initiates a chain of trust during the different boot stages.

4.1 CHAIN OF TRUST

The <t-base security relies on the principle of the chain of trust. The chain of trust starts with an immutable software component in the system which is the secure anchor of the boot process that is implicitly trusted.

The secure anchor is the root of the chain of trust and contains information which is required to verify the next link in the chain of trust. This transitive process is applied recursively from one link to the next.

In an ARM TrustZone enabled SoC the secure boot ROM (see *chapter 3.2*) represents the secure anchor.

4.2 BOOT STAGES

The secure boot process can be divided into three logical boot stages (BS0 to BS2). The reset behaviour of SoC is design specific. To grant the correct set up of the chain of trust the boot process must start with executing the secure anchor BS0.

It is worth mentioning that BS1 und BS2 may be split up or merged in practice depending on the infrastructure which is set up by the SiP. Nevertheless the logical boot stages apply.

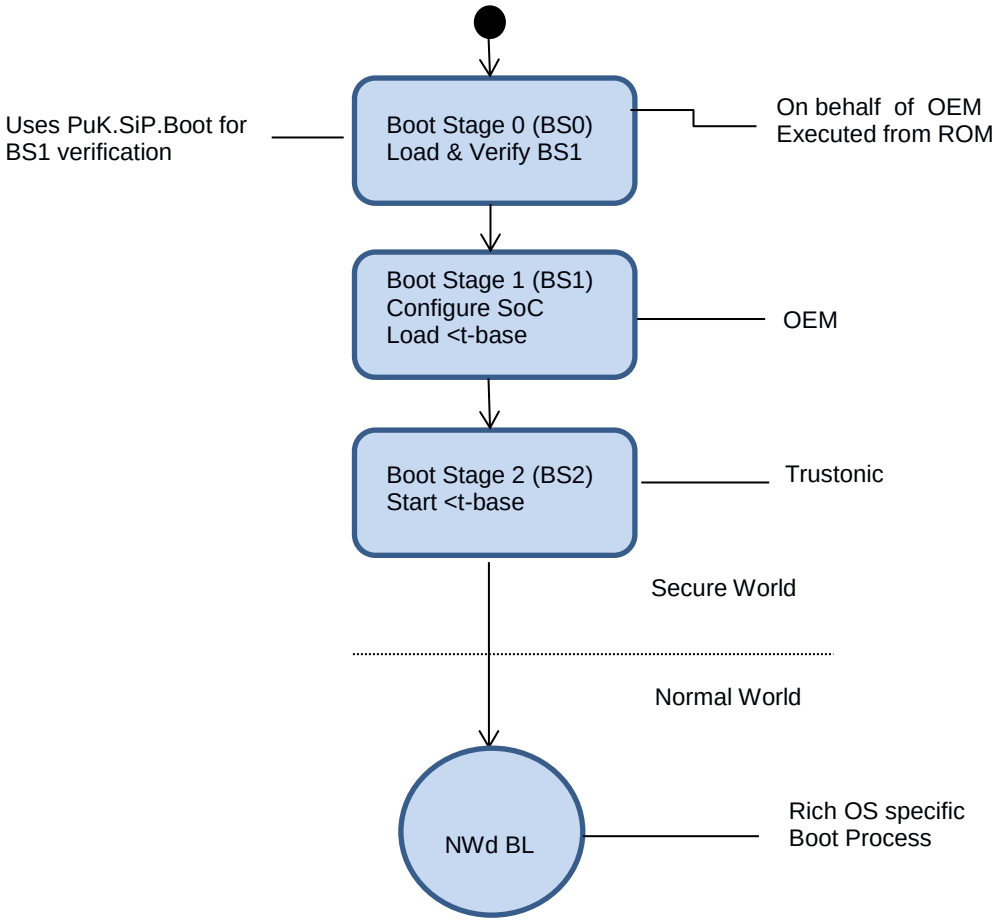


Figure 3: Boot Process Overview.

4.2.1 Boot Stage 0 (BS0) – Load and Verify BS1

After reset boot stage BS0 has immediately to be executed as the secure anchor for the chain of trust. BS0 contains code to load and verify the boot stage BS1. All code and data needed for verification must be loaded from the secure boot ROM of the SoC.

The SoC Boot ROM contains the boot code of Boot Stage BS0.

During Boot Stage BS0 the code of Boot Stage BS1 is verified with SiP's Public Key (PuK.SiP.Boot).

4.2.2 Boot Stage 1 (BS1) – Configure SoC and Load <t-base

In boot stage BS1 the configuration of the SoC is performed. Peripherals as well as the Trust Zone Protection Controller (TZPC) (see chapter 3.1) are set up according to the security requirements of <t-base TEE.

Boot stage 1 enforces the isolation of NWd and SWd by setting up the TZPC.

After the SoC hardware has been set up, boot stage BS1 loads the <t-base Boot image which has been verified as part of the BS1 verification.

4.2.3 Boot Stage 2 (BS2) – Start <t-base and NWd OS

Once <t-base is loaded in BS1, BS2 starts <t-base TEE. Once <t-base is initialized, <t-base switch the system in non-secure mode and starts the Normal World bootloader.

5 <T-BASE TEE SECURITY

This chapter assumes that the device and <t-base has securely booted as described in the previous chapter *4.*

The security mechanisms of <t-base are based on several components including secure SoC peripherals.

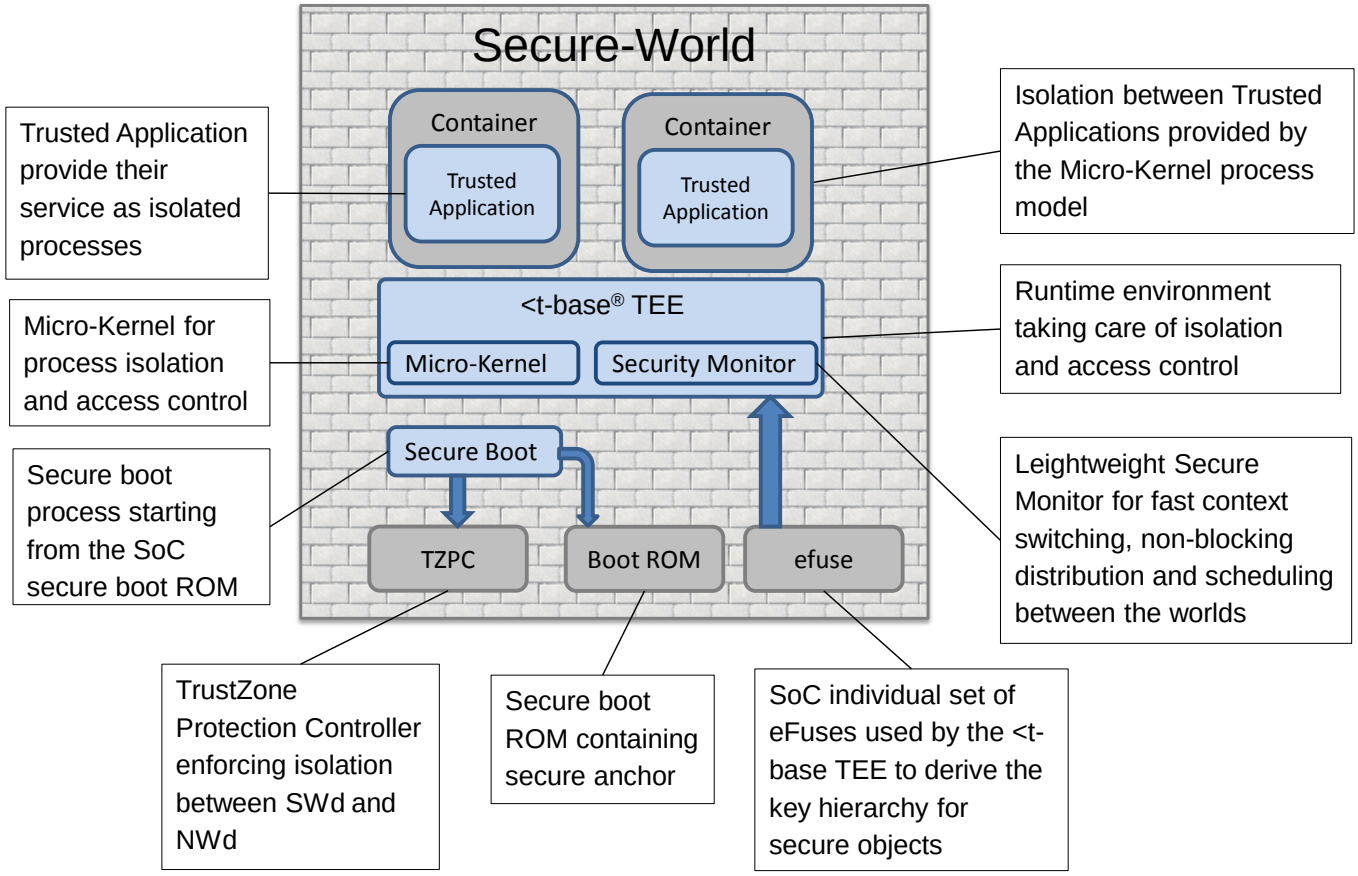


Figure 4: <t-base SWd components.

5.1 ISOLATION

5.1.1 NWd - SWd Isolation

During the secure boot process <t-base configures the SoC in a way that software executed in the NWd cannot access peripherals and data dedicated to the SWd. This is typically achieved by configuring the SoCs Trust Zone Protection Controller TZPC before control is

handed over to any software component considered to have a lower level of trust as <t-base.

5.1.2 Intra SWd Isolation

The <t-base micro-kernel together with an MMU provides process isolation based on traditional kernel techniques like separated address spaces, pre-emptive multitasking and inter-process communication. In addition the <t-base kernel implements extended features to prevent denial-of-service attacks like resource grabbing or service blocking.

Each time a TA is started, the <t-base TEE defines the capabilities of that TA. The <t-base kernel ensures compliance with these capabilities. The capabilities of a <t-base TA are comprised of memory access rights, execution priority, resource usage and inter-process communication rights.

Each process is executed in a separated address space. Private data of a TA is not accessible by another TA, except the <t-base TEE set-ups shared memory between two or more address spaces. Illegal memory access is intercepted by the <t-base kernel and reported to the <t-base TEE.

5.2 ACCESS CONTROL

Secure peripherals and TrustZone runtime have to be protected against illegal usage. Therefore <t-base enforces strict access control mechanisms.

5.2.1 NWd to SWd Access Control

NWd applications are not able to access secure peripherals directly due to the isolation principle explained in section 5.1.1. Additionally access to the code execution resources of the TEE is protected by the security scheme described in chapter 5.1.1. Only service providers certified by <t-directory are able to provide TAs which can be loaded and executed by <t-base.

5.2.2 Intra SWd Access Control

Secure peripherals are exclusively controlled by the <t-base TEE. In order to avoid resource conflicts <t-base applies an access control policy to the usage of resources. TAs can use secure peripherals via the TA API only.

5.3 <T-BASE DATA PROTECTION

An implementation of <t-base also requires a secure NVM (e.g. an eFuse, see Fig. 6) which is used during the Device Binding operation to bind a first cryptographic key to the SUID, to support the Content Management commands.

The <t-base TEE incorporates a SWd exclusive cryptographic library. This exclusive cryptographic library may be accessed by system and service provider TAs via the <t-base TA API for the implementation of their security functions.

Depending on the SoC, the implementation of the cryptographic library may be based on secure crypto peripherals, like hardware cryptographic co-processors.

The Content Management TAs uses the functionality of that cryptographic library to secure the loading and installation of TA code.

<t-base supports dynamic loading of TA code. It is crucial for the overall system security that only System TAs approved by Root or TAs of approved service providers can be executed by <t-base system (see chapter *<t-base Content Management*).

<t-base does not require direct access to secure non-volatile storage. Instead it offers a mechanism to wrap data into Secure Objects, which contain the data in encrypted and integrity protected form. Secure Objects are passed by the TAs to the Rich OS client application and can be stored in an arbitrary insecure location.

When the data is needed again, the Secure Object must be returned to the TA for the unwrapping of data.

Every Secure Object is encrypted with a key derived from the device master key K.Device.Fuse and other specific context data. For every TA, a unique key is derived, and within the TA, different keys can be derived, too. The keys are managed by <t-base and cannot be exported. This ensures that any wrapped data cannot be un-wrapped by any other party besides the platform that has generated it.

5.4 TA EXECUTABLES

There are two types of TAs in <t-base:

- Service Provider TAs (SP TAs).
- System TAs (for example Content Management TA, ...)

5.4.1 Service Provider TA

There is exactly one Service Provider TA (code) assigned to one TA Container.

SP-TAs can be installed on-demand after the registration of referenced TA Container. The Open Session command shall check the corresponding TA Container for Confidentiality, Authenticity.

UUID is the identifier of the TA Container where the symmetric key K.SP.TltEnc for TA code decryption and HMAC verification shall be stored.

Version is the version number of the TA binary.

5.4.2 System TA

A System TA must always be installed at the same time with the <t-base Boot Image and is subject to other preconditions as a SP TA.

A System TA:

- Has no reference to a TA Container
- Has a signature (created with the method RSASSA-PSS [PKCS-1])
- Contains a public key (PuK.Vendor.TltSig) to verify the signature

5.5 SECURE OBJECTS

5.5.1 Overview

As <t-base system and TA data shall be persistent beyond a TA session, <t-base provides the Secure Object mechanism to store data in the NWd.

Secure Objects provide authenticity, integrity protection and encryption for data which can then be stored in the NWd.

The Secure Object API provides integrity, confidentiality and authenticity for data that is sensitive but needs to be stored in not trusted NWd memory. Secure objects provide device binding. Respective objects are only valid on a specific device.

The user of the Secure Object mechanism may choose, which data are to be stored in plaintext (but integrity protected) and which data are to be stored encrypted and integrity protected.

Data Integrity

A Secure Object contains a message digest (HMAC) that ensures data integrity of the user data. The HMAC value is computed and stored during the *tlApiWrapObject()* operation (before data encryption takes place) and recomputed and compared during the *tlApiUnwrapObject()* operation (after the data has been decrypted).

Confidentiality

Secure objects are encrypted with AES CBC using context-specific 256-bit keys that are never exposed, neither to the normal world, nor to the TA. It is up to the user to define how many bytes of the user data are to be kept in plain text and how many bytes are to be encrypted. The plain text part of a Secure Object always precedes the encrypted part.

Authenticity

As a means of ensuring the trusted origin of Secure Objects, the *tlApiWrapObject* () operation stores the TA ID (SPID, UUID) of the calling TA in the Secure Object header (as Producer). This allows TAs to only accept Secure Objects from certain partners. This is most important for scenarios involving secure object sharing.

Protection against Replay Attacks

<t-base does not protect Secure Objects against replay attacks. Is such protection is needed, it must be provided by other means.

5.5.2 Secure Object Format

A Secure Object looks like this

Table 1: Secure Object Format.

Group	Field	Type
Header	Type	uint32

	SO Version	uint32
	Context	uint32
	Lifetime	uint32
	Producer	uint8[20]
	PlainLength	uint32
	ToBeEncryptedLength	uint32
User data	Plaintext data	uint8[PlainLength]
	Data to be encrypted	uint8[EncryptedLength]
Appendix	HMAC	uint8[32]
	ORng	uint8[16]

5.5.2.1 Secure Object Type

<t-base differentiates two Secure Object types:

- MC_SO_TYPE_REGULAR: the regular secure object type.

5.5.2.2 Secure Object Version

The Secure Object version (SO version) is administrated by the <t-base system. The version is used to allow the Secure Object subsystem to discard old secure objects.

5.5.2.3 Secure Object Context

The concept of context allows for sharing of Secure Objects. For the time being there are three kinds of context:

- MC_SO_CONTEXT_TLT: TA context. The secure object is confined to a particular TA. This is the standard use case.
- PRIVATE WRAPPING: If no consumer was specified, only the TA that wrapped the Secure Object can unwrap it.
- DELEGATED WRAPPING: If a consumer TA is specified, only the TA specified as 'consumer' during the wrap operation can unwrap the Secure Object. Note that there is no delegated wrapping with any other contexts.
- NOTE: in Secure Object format before 2.1, wrapping in System TA using MC_SO_CONTEXT_TLT did not consider UUID, thus behaving similarly to MC_SO_CONTEXT_SP. In format 2.1 and later System TA MC_SO_CONTEXT_TLT wrapping works similarly to service provider TA wrapping.

- MC_SO_CONTEXT_SP: Service provider context. Only TAs that belong to the same Service Provider can unwrap a secure object that was wrapped in the context of a certain service provider. System TAs are considered to belong to same SP.
- MC_SO_CONTEXT_DEVICE: Device context. All TAs can unwrap secure objects wrapped for this context.

5.5.2.4 Secure Object Lifetime

The concept of a lifetime allows limiting how long a Secure Object is valid. After the end of the lifetime, it is impossible to unwrap the object.

Three lifetime classes are defined:

- MC_SO_LIFETIME_PERMANENT: Secure Object does not expire.
- MC_SO_LIFETIME_POWERCYCLE: Secure Object expires on reboot.
- MC_SO_LIFETIME_SESSION: Secure Object expires when TA session is closed. The secure object is thus confined to a particular session of a particular TA. Note that session lifetime is only allowed for private wrapping in the TA context MC_SO_CONTEXT_TLT.

5.5.2.5 Producer

The Producer is the TA ID (SPID, UUID) of the TA, which called the wrapping function *tlApiWrapObject()* to construct that specific Secure Object. The Producer parameter is maintained by <t-base.

As a means of ensuring the trusted origin of Secure Objects, the tlApiWrapObject() operation stores the TA ID (SPID, UUID) of the calling TA in the Secure Object header (as Producer). This allows TAs to only accept Secure Objects from certain partners. This is most important for scenarios involving secure object sharing.

6 KEYS IN THE <T-BASE SYSTEM

This section describes the difference cryptographic keys used in <t-base

6.1 DEVICE MASTER KEY K.DEVICE.FUSE

This device individual symmetric master key (32 bytes) shall be generated for each device and is only known and accessible by SWd.

Table 2: Device Master Key K.Device.Fuse.

Name	Origin	Used by	Environment
K.Device.Fuse	Device	<t-base	SWd

The key K.Device.Fuse is used as master key for context key derivation (see chapter 6.5). The derived context keys K.Device.Ctxt are used to protect the Context dependent Secure Objects stored in NWd.

The storage of the master key K.Device.Fuse is highly dependent on the underlying SoC platform. Not all platforms allow storing the 256 bit AES key K.Device.Fuse in a write once fuse of the SoC.

This chapter describes for such platforms, how K.Device.Fuse is derived from the immutable 12 byte-length Chip Serial Number (CSN) and an immutable Device SEED (16 to 32 byte) only accessible by SWd.

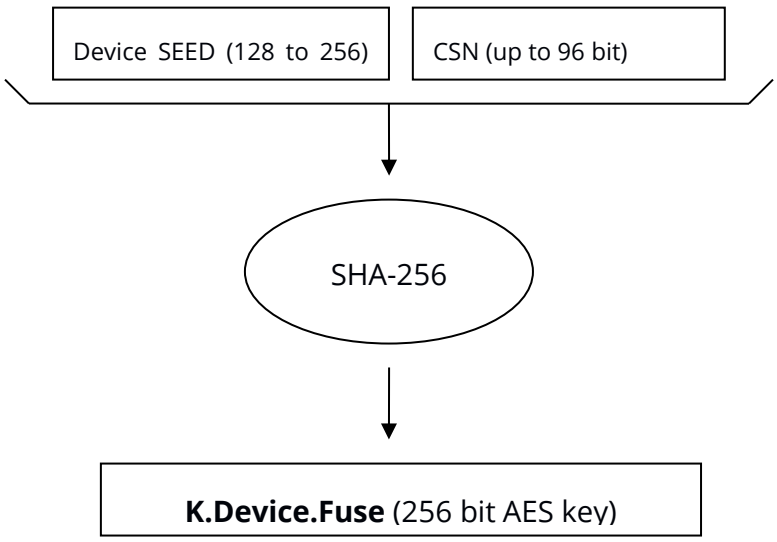


Figure 5: Derivation of K.Device.Fuse.

The Device SEED and the CSN are concatenated; the resulting string (28 to 44 byte-length) shall be hashed with hash algorithm SHA-256. The resulting 256 bit hash value is used as the master key K.Device.Fuse.

Note:

The method and modality to get the invariable Device SEED is platform-specific as well as the length of this element (16 to 32 byte), but all read operations of Device SEED can only be accessible in context of SWd.

6.2 DEVICE BINDING KEYS

For Device Binding see the chapter *Device Binding*.

The Device Binding is a process defined outside the <t-base itself and is dependent on the infrastructure of the OEM. <t-base provides some commands to support Device Binding.

Following keys are only examples and are to be used for protecting the communication exchanges between Trustonic and the OEM.

Table 3: Device Binding Keys.

Name		Origin	Used by	Environment
PkP.Vendor.MCAct	PrK.Vendor.MCAct	Vendor	Root	Root
	PuK.Vendor.MCAct		KPH	OEM
PkP.Vendor.Receipt	PrK.Vendor.Receipt	Vendor	KPH	OEM
	PuK.Vendor.Receipt		TEE Vendor	TEE Vendor
PkP.Root.Transport	PrK.Root.Transport	Root	Root	n.a.
	PuK.Root.Transport		KPH	OEM
PkP.Kph.Request	PrK.Kph.Request	KPH	KPH	OEM
	PuK.Kph.Request		<t-base OS	OEM

6.2.1 <t-base Activation Key – PkP.Vendor.MCAct

PrK.Vendor.MCAct is handed over by the TEE Vendor to the Root in a secure manner. It is used to decrypt the ED.SUID.AuthToken from the SD.Receipt.

PuK.Vendor.MCAct is stored in the KPH. It is used to encrypt the SUID.

6.2.2 Root Transport Key – PkP.Root.Transport

The Root key PuK.Root.Transport encrypts the SoC individual key K.SoC.Auth.

The encrypted SoC Authentication Key K.SoC.Auth can only be decrypted by Root with PrK.Root.Transport

6.2.3 PkP.Vendor.Receipt

PuK.Vendor.Receipt is used by the TEE vendor to verify the receipt which was signed by the KPH

PrK.Vendor.Receipt is used by the KPH to digitally sign the receipts.

6.2.4 PkP.Kph.Request

Modulus bit length is 2048, the value of the public Exponent shall be 3

PuK.Kph.Request is used by the <t-base to verify the request for the authentication token during the device binding process.

PrK. Kph.Request is used by the KPH to sign the GenerateAuthToken command sent to the <t-base.

6.3 AUTHENTICATION KEYS

The Authentication keys are used during the authentication operations SOC AUTHENTICATION, ROOT AUTHENTICATION and SP AUTHENTICATION (see chapter *Content Management Operations*). These keys are transferred to the system during DEVICE BINDING (chapter *Device Binding*), ROOT REGISTRATION (chapter 8.2.3) and SP REGISTRATION (chapter 8.2.7).

Table 4: Authentication keys.

Name	Origin	Used by	Environment
K.SoC.Auth	KPH	<t-base, Root	SWd, Root
K.Root.Auth	Root	<t-base, Root	SWd, Root
K.SP.Auth	SP	<t-base, SP	SWd, SP

6.3.1 K.SoC.Auth

This symmetric key is generated by the KPH in the OEM context. The pair K.SoC.Auth and SUID (SoC Unique Identifier) shall be transferred to Root (SD.Receipt) for the SOC AUTHENTICATION (see chapter *Content Management Operations*) later.

K.SoC.Auth is used during SOC AUTHENTICATION to negotiate the session keys SKi and SKc (see chapter 6.4).

The K.SoC.Auth shall be stored in Secure Object SO.AuthToken.

6.3.2 K.Root.Auth

This symmetric key is generated by Root. The Root has to authenticate to the Device with this key (ROOT AUTHENTICATION, see chapter *Content Management Operations*).

K.Root.Auth is used during ROOT AUTHENTICATION to negotiate the session keys SKi and SKc (see chapter 6.4).

The K.Root.Auth shall be stored in the Secure Object SO.RootCont.

6.3.3 K.SP.Auth

This symmetric key shall be generated by Service Provider.

The Service Provider has to authenticate to the Device with this key (SP AUTHENTICATION, see chapter*Content Management Operations).*

K.SP.Auth is used during SP AUTHENTICATION to negotiate the session keys SKi and SKc (see chapter 6.4).

The K.SP.Auth shall be stored in Secure Object SO.SPCont.

6.4 SESSION KEYS FOR CONTENT MANAGEMENT SESSIONS

Note: see chapter 7.2 on Content Management Session and chapter *Content Management Operations* for the Authentication operations.

For the negotiation of session keys see the Appendix.

Note:

The naming Ski, SKc and SSC are shortcuts for the session key negotiation depending on the authentication operation (SOC AUTHENTICATION, ROOT AUTHENTICATION and SP AUTHENTICATION. The full names are:

- K.Device.SKi, K.Device.SKc and Device.SSC
- K.Root.SKi, K.Root.SKc and Root.SSC
- K.SP.SKi, K.SP.SKc and SP.SSC

6.5 CONTEXT KEYS FOR SECURE OBJECT PROTECTION

The Secure Objects of a Context must be stored in NWd and therefore protected with a Context derived key (K.Device.Ctxt) so that exclusive data are protected against the other contexts with the exception of shared keys and data.

Those context keys are derived from K.Device.Fuse every time those keys are needed. The context keys are never stored persistently.

Note:

The naming K.Device.Ctxt used here is a shortcut for the context specific derived keys

- K.Device.CtxtSys
- K.Device.CtxtSP

- K.Device.CtxtTLT.

Note:

The key naming K.Device.Ctxt here shall be a shortcut for the context specific derived keys K.Device.CtxtSys, K.Device.CtxtSP and K.Device.CtxtTLT.

6.6 SERVICE PROVIDER TA CODE ENCRYPTION KEY

Table 5: TA Code Encryption Key.

Name	Origin	Used by	Environment
K.SP.TltEnc	SP	<t-base, SP	SWd, Root

The TA Code Encryption Key is a Service Provider key used to encrypt / decrypt the Service Provider's TA binary code.

This symmetric key is generated by Service Provider. The Service Provider has to encrypt its TAs with K.SP.TltEnc.

The key K.SP.TltEnc is transferred during the operation TA REGISTRATION (chapter 8.3.5) and stored in the TA Container Secure Object SO.TrusletCont.

6.7 <T-BASE SYSTEM TA SIGNATURE KEY

<t-base System TAs, like the Content Management TA, are TEE Vendor or OEM specific TAs, which are already pre-installed in <t-base. System TAs are signed with PrK.Vendor.TltSig and signatures are verified with PuK.Vendor.TltSig During the loading of a System TA for execution in Secure RAM, the System TA signature is verified.

Table 6: System TA Signature Keys.

<table>
<tr><th colspan="2">Name</th><th>Origin</th><th>Used by</th></tr>
<tr><td rowspan="2">PkP.Vendor.TltSig</td><td>PrK.Vendor.TltSig</td><td rowspan="2">Vendor</td><td rowspan="2">Vendor</td></tr>
<tr><td>PuK.Vendor.TltSig</td></tr>
</table>

6.7.1 PuK.Vendor.TltSig

PuK.Vendor.TltSig is used by <t-base to verify the signature from a System TA. PuK.Vendor.TltSig is a part of a System TA. Its Hash value (SHA-256) shall be calculated by <t-base loader, finally the hash value shall be compared with the Hash value stored in <t-baseImage before this TA is started.

6.7.2 PrK.Vendor.TltSig

PrK.Vendor.TltSig is used by TEE Vendor to create a signature for a System TA.

7 <T-BASE CONTENT MANAGEMENT

The Trusted Applications of <t-base can be managed Over The Air through a Content Management Protocol which includes the management of Service Providers and the management of the Trusted Applications for each Service Provider.

Those Root administrative operations which are described in the chapter *Root Administrative Operations* are:

- Registration and activation of Service Providers
- Locking and unlocking of Service Providers
- Deletion of Service Providers

The Service Provider operations which are described in the chapter Service Provider Administrative Operations are:

- Registration and activation (installation) of Service Provider TAs
- Personalization of TAs
- Locking and unlocking of TAs
- Deletion of SP TAs.

The entities involved in the <t-base Content Management are the respective backend systems (BE) of Root or Service Providers and the SWd Content Management TA (CMTA) on device.

A secure communication mechanism is established between the respective backend systems of Root or SP and the CMTA on the <t-base enabled device (see Authentication Operations).

The Content Management TA defines the Content Management Protocol (CMP), which comprises of the supported Content Management commands and the policies of usage of that commands. The CMP enables the implementation of the content management operations of Root and SP.

For each approved Service Provider a so called SP Container is established by Root on the device. Service Providers may install their SP TA Containers into their respective SP Containers (see also *Content* Objects).

7.1 CONTENT MANAGEMENT OPERATIONS OVERVIEW

In <t-base content management comprises of several operations described in detail in chapter Content Management .

The following entities are involved in the content management operations:

- <t-directory (Root) and Service Provider (SP) backend. Those backend systems communicate via the NWd application Provisioning Agent (PA) to the Content Management TA (CMTA).

- Provisioning Agent (PA): An NWd application which is the link component between the backend systems of Root and SP, and the SWd Content Management TA (CMTA). How the PA communicates with the backend systems is beyond the scope of this document. The PA forwards the content management commands from the backend systems to CMTA.
- The SWd system Content Management TA (CMTA) which provides the Content Management Protocol (CMP) consisting of the content management commands and their semantics.

The content management operations are divided in following groups:

7.1.1 Device Binding

Device Binding is not really a Content Management operation, but is the central prerequisite operation, which prepares the functionality of the Content Management. After Device Binding a first SoC Authentication (see *Authentication Operations*) is possible.

Device Binding is the process of associating a device SoC with a cryptographic key. The device is identified by the SoC Unique Identifier (SUID) and paired with a cryptographic key in production system of the *OEM*. The SUID of the SoC serves as unique identifier of the device; the key which is bound to that SUID enables Root to establish a first secure communication channel to that device, to drive further Root administrative content management processes (see *Root Administrative Operations*). The SUID and the SoC Authentication Key K.SoC.Auth are transmitted as Authentication Token Binding from the production site of the OEM to the backend system of Root. In further content management processes the Authentication Token is used by <t-directory to identify the device in the <t-base ecosystem.

Note:

This Device Binding process is repeatable.

7.1.2 Authentication Operations

Authentication operations are used to authenticate <t-directory (Root) and the Service Provider against the device and vice versa (mutual authentication). The authentication operations establish session keys and create a secure channel for the Root and Service Provider administrative commands.

7.1.3 Root Administrative Operations

Root Administrative operations are administrative commands issued by Root within a secure channel after successful Root Authentication.

These operations allow Root to

- Register and deregister the Root Container
- Lock and unlock the Root Container
- Register, register-activate, activate and deregister a Service Provider Container
- Lock and unlock a Service Provider Container

7.1.4 Service Provider Administrative Operations

Service Provider administrative operations (see chapter 8.3) are administrative commands issued by the Service Provider within a secure channel after successful SP Authentication.

These operations allow the Service Provider to

- Activate the SP Container
- Lock and unlock the SP Container
- Register, register-activate, activate and deregister a TA Container
- Personalize the TA Container
- Lock and unlock a TA Container

7.2 CONTENT MANAGEMENT SESSION

Content management operations require a secure channel between the backend system of Root or a Service Provider and the Content Management TA in <t-base. The secure channel provides mutual authentication of the entities and integrity and confidentiality of the transmitted data. The secure channel between those entities is called the Content Management session.

A content management session starts with an authentication operation (see *Content Management Operations)* and administrates Security States.

The authentication operations create a secure channel in which the content management commands are transmitted securely between the backend systems of Root or Service Provider and the Content Management TA.

A content management session ends after:

- An erroneous command is sent (unknown CM command, error in the processing of the command)
- A cryptographic error in the secure channel session
- A command is sent which is not cryptographically secured by secure messaging. In this case the content management session is closed (security state is set to AUTH_NONE) before the command is processed (e.g. GetSUID, GetVersion).
- A content management session termination by the command AuthenticateTerminate
- The termination of the TA session
- A reset of the device.

A content Management session shall administrate the following data in volatile memory of <t-base:

- Service Provider ID (SPID)
- Session keys SKc, SKi (see chapter 6.4) and Send Sequence Counter SSC (see chapter 6.4)
- Security State

Note:

All data invoked in a content management session shall be maintained in secure volatile memory.

For security reasons the above volatile data must be initialized and cleared by the Content Management TA when a Content Management Session is started or terminated.

7.3 SECURITY STATES

The Content Management TA administrates the following security states of the content management sessions in <t-base internal volatile memory associated to the Authentication operations:

- AUTH_NONE
- AUTH_SOC
- AUTH_ROOT
- AUTH_SP

The transition from one security state to the other is achieved by the processing of the authentication commands, a device reset, an error in the processing of the secure messaging command, or the end of a Content Management session.

7.3.1 Security State AUTH_NONE

No administrative commands (Root and Service Provider administrative commands) can be executed in the security state AUTH_NONE.

This security state is the default security state after the reset of <t-base, a cryptographic error in the secure channel processing, an explicit session termination (AuthenticateTerminate) or when the authentication commands fail.

7.3.2 Security State AUTH_SOC

The security state AUTH_SOC allows executing the Root administrative command RootContRegisterActivate.

Security State AUTH_SOC is achieved from any security state by successful processing of commands BeginSoCAuthentication and Authenticate.

7.3.3 Security State AUTH_ROOT

The security state AUTH_ROOT is the root administrative security state. In this state Root is allowed to initiate Root administrative commands.

Security State AUTH_ROOT is achieved from any security state by successful processing of the authentication commands BeginRootAuthentication and Authenticate.

7.3.4 Security State AUTH_SP

The SP administrative commands can only be executed in the Security State AUTH_SP.

The security state AUTH_SP is reached from any security state by successful processing of the authentication commands BeginSPAuthentication and Authenticate.

7.3.5 Security State Transitions

Security State transitions are achieved by mutual authentication using the appropriate authentication commands a device reset, a cryptographic error in the secure messaging processing or a secure messaging termination by an explicit command.

The following figure illustrates possible security state transitions:

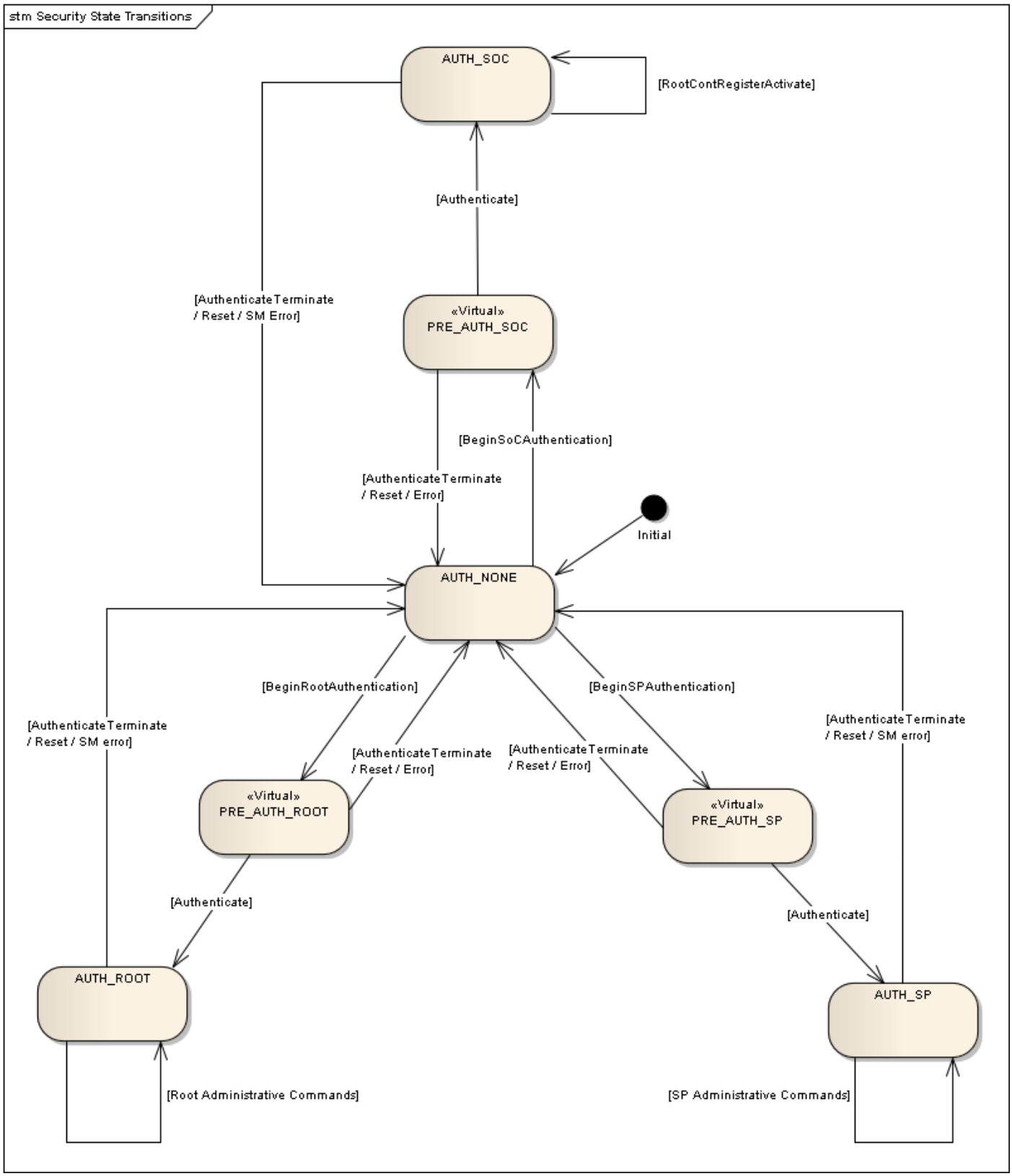


Figure 6: Security State Transitions.

After device reset, the initial security state of <t-base is AUTH_NONE. When <t-base is not yet activated (directly after Device Binding) only the Authentication Token with K.SoC.Auth is known by <t-base.

In this case only Root knowing that Authentication Token is able to successfully authenticate to <t-base with SoC authentication. In the security state AUTH_SOC, Root issues the command RootContRegisterActivate and creates the Root Container containing the Root authentication key K.Root.Auth. The Authentication Token is deleted and no further SoC Authentication is possible.

After Root have been registered and activated, Root is able to authenticate itself to <t-base with Root authentication using the Root authentication key K.Root.Auth. In the security state AUTH_ROOT Root is able to issue Root administrative commands.

When a SP Container had been registered and activated by Root, the Service Provider is able to authenticate itself to its SP Container using SP authentication with SP Authentication key K.SP.Auth. In security state AUTH_SP, the authenticated SP is able to issue SP administrative commands.

When the device is reset, Root or SP issues an AuthenticateTerminate _command, a TA session ends or a cryptographic error occurs during the processing of the secure messaging, the security state is reset to AUTH_NONE.

7.4 CONTENT OBJECTS

The organization of managed content is a hierarchical structure of Content Objects. The Content objects are internal data structures, which need to be stored in the NWd <t-base registry.

Table 7: Types of Content Objects.

Object type	**Cardinality**
Authentication Token	After Device Binding one Authentication Token
Root Container	After Root Registration Activation one Root Container replacing the Authentication Token.
Service Provider Container	A Root Container may reference up to 16 SP Containers, which are created during SP REGISTRATION.
TA Container	A SP Container may reference up to 16 TA Container, which are created during TA REGISTRATION.
SP TA	Each SP TA is associated to a TA Container of that SP.
System TA	A System TA is **not** associated to a TA Container. System TAs are included in the <t-base boot image. At least the Content Management TA (CMTA) is available.

	<t-base may support several other System TAs.
TA Personalization Data	After a TA Container is registered, that TA Container may be personalized with 0, 1 or more Personalization Data objects.

The following figure illustrates the content object hierarchy as a tree-like diagram; the tree nodes are content objects, the edges denote the operation that yields a new content object:

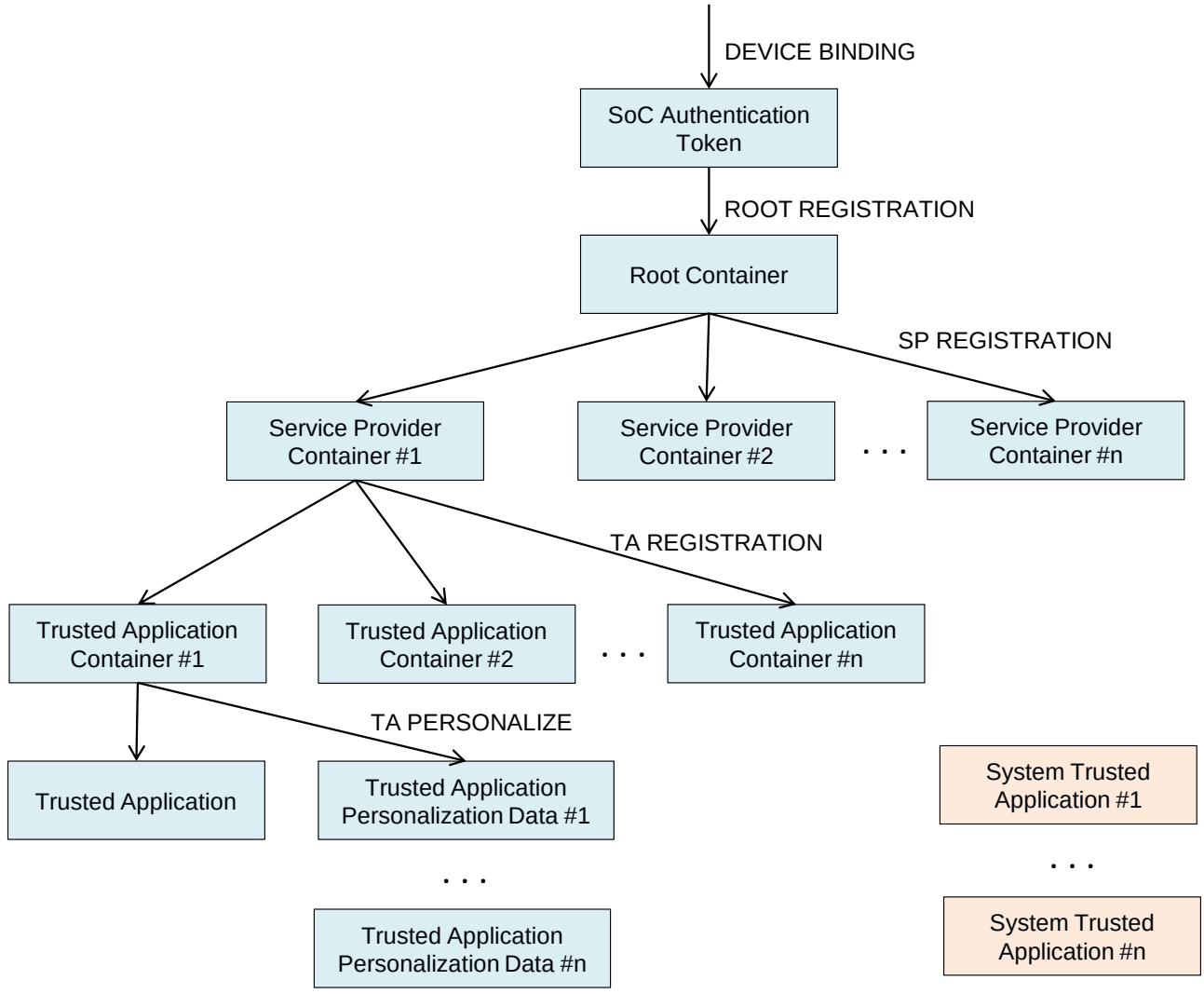


Figure 7: Content Object Hierarchy.

7.5 CONTAINER LIFE CYCLE STATES

For the container types Root, SP and TA Container, a life cycle state is associated. The Content Management administrative commands change the affiliated container life cycle states. The following chapters describe the Root Container, Service Provider Container and TA Container life cycle states and their respective state transitions.

7.5.1 Generic Container Life Cycle States

<t-base differentiates between Root, SP and TA Containers. Each container is a life cycle state associated. <t-base knows following generic container life cycle states.

Table 8: Generic Container Life Cycle States.

Container Life Cycle State	Description
MC_CONT_STATE_REGISTERED	When a container gets into life (is registered) the respective container object is created and its life cycle state is set to MC_CONT_STATE_REGISTERED. A container in this life cycle state is not fully functional. It may be personalized and an additional activation step is required. **Note**: Not all container types reach this intermediate life cycle state. The Root Container is registered and activated in one step.
MC_CONT_STATE_ACTIVATED	This state is the fully operational life cycle state of a container. Each allowed content management command appropriate for that specific container type may be issued in this state. This life cycle state of a container may be reached of its registered state or directly out of void by a combined ...RegisterActivate... command where appropriate.
MC_CONT_STATE_ROOT_LOCKED	Due to business or security reasons Root may decide to lock its Root Container or to lock one of the associated SP Containers. The life cycle state of that Container is changed to MC_CONT_STATE_ROOT_LOCKED. A Container in that life cycle state may only be unlocked by the appropriate Root administrative command and returns back to the previous life cycle state it had before the lock occurred.

MC_CONT_STATE_SP_LOCKED	Due to business or security reasons a Service Provider may decide to lock its SP Container or one of its TA Containers. The life cycle state of that Container is set to MC_CONT_STATE_SP_LOCKED. A Container in that life cycle state may only be unlocked by appropriate SP Administrative command and returns to the previous life cycle state it had before the lock occurred.
MC_CONT_STATE_ROOT_SP_LOCKED	This state may only be valid for a SP Container. After SP had decided to lock its SP Container (MC_CONT_STATE_SP_LOCKED), Root may decide due to business or security reasons to lock that SP Container too. The life cycle state of that container is changed to MC_CONT_STATE_ROOT_SP_LOCKED. In this state first Root needs to unlock that SP Container, before the Service Provider is able to unlock its SP Container.

Note:

Not all described generic life cycle states are allowed for each container type. The allowed life cycle state transitions of a container type are described in the following sub-chapters.

7.5.2 Root Container Life Cycle State Transitions

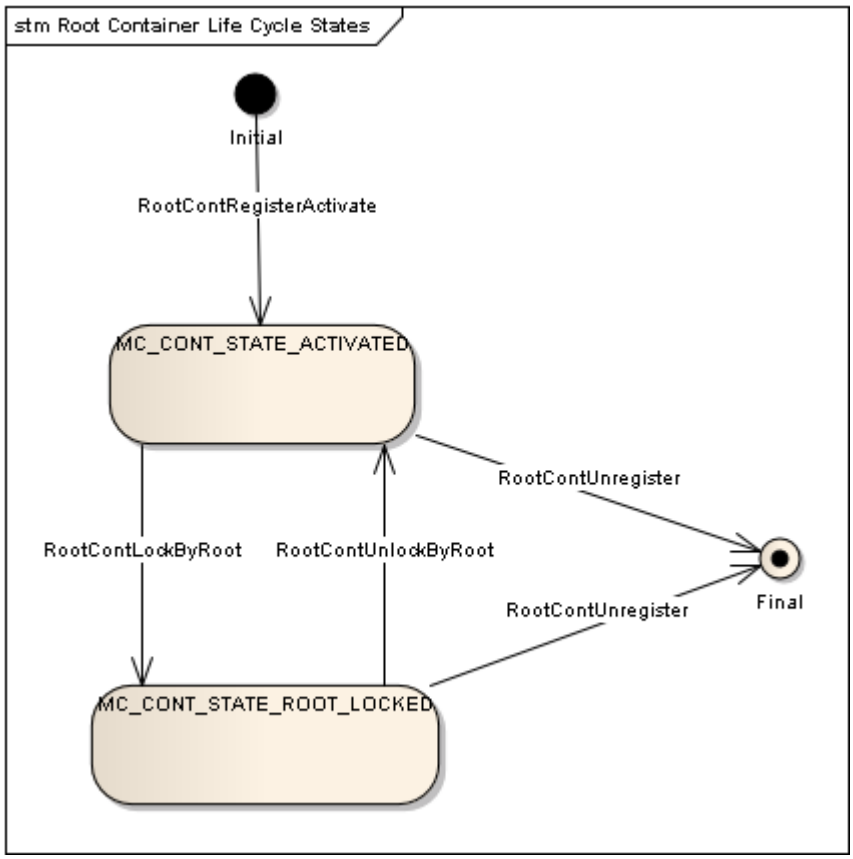


Figure 8: Root Container Life Cycle State Transitions.

A Root Container is registered and activated in one step by the Root administrative command RootRegisterActivte. In this step the Root Container reaches its fully functional state MC_CONT_STATE_ACTIVATED immediately.

Due to business or security reasons Root may decide to lock its Root Container by issuing the command RootContLockByRoot. The life cycle state of that Root Container changes to MC_CONT_STATE_ROOT_LOCKED.

Root Container lock does not take any effect on the Root Container itself. That means the Root Container is still fully functional. The Root Container lock takes effects on all SP Containers and TA Containers in the sub-hierarchy below the Root Container.

When the Root Container is locked, all dependent SP Container and TA Container in the sub-hierarchy of the Root Container are implicitly locked too (on explicit and implicit container locks see chapter Implicit and Explicit Container Locks). That means after a Root lock no further SP administrative commands may be issued.

Only Root may unlock its Root Container by issuing RootContUnlockByRoot command. The Root unlock also unlocks the implicitly locked dependent SP and TA Container back to their state they had before the Root lock.

In each valid Root Container life cycle state the command RootContUnregisterRoot may be issued by Root. This command deletes and frees the Root Container and all dependent SP

and TA Containers. <t-base is inactivated and changes its general state back to the state it had directly after Device Binding.

7.5.3 SP Container Life Cycle State Transitions

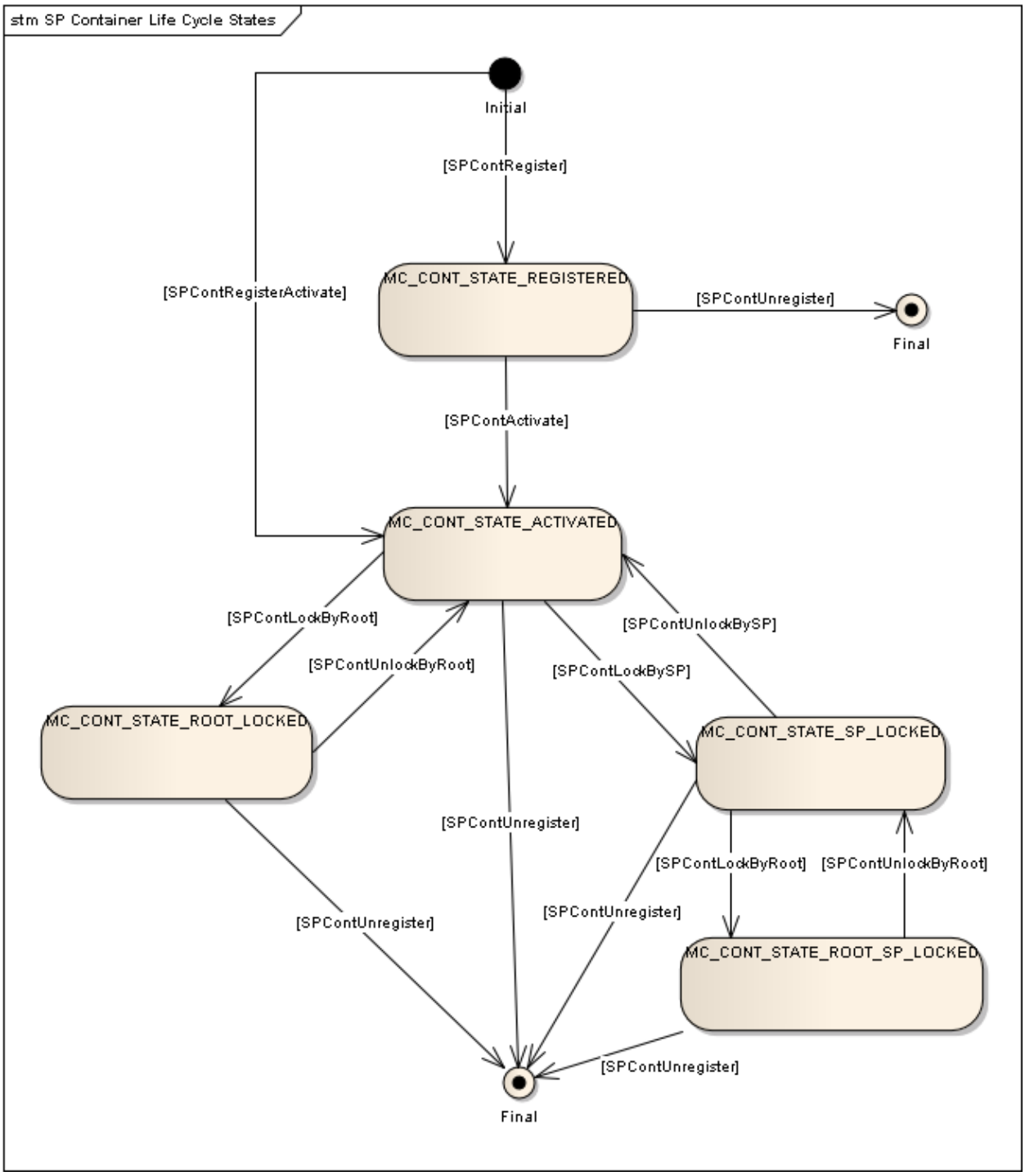


Figure 9:SP Container Life Cycle State Transitions.

A SP Container is created with the Root administrative command SPContRegister. The life cycle state is set to MC_CONT_STATE_REGISTERED. SPContActivate brings the SP Container in its fully functional state MC_CONT_STATE_ACTIVATED.

Due to business or security reasons Root may decide to lock the SP Container with SPContLockByRoot (life cycle state MC_CONT_STATE_ROOT_LOCKED). Locking of the SP Container means that all dependent TA Container, which are not explicitly locked gets implicitly locked (see Implicit and Explicit Container Locks).

When the SP Container is locked by Root, the Service Provider is not able to authenticate to that respective SP Container and to issue SP administrative commands. Only Root is able to unlock the Root locked SP Container by issuing command SPContUnlockByRoot.

In the activated life cycle state the SP may lock its SP Container with the SP administrative command SPContLockBySP.

The SP Container Lock by SP does not take direct effect on the SP Container that means all SP administrative commands may be issued to the SP Container. But the dependent TA Container of that SP gets implicitly locked, when not already explicitly locked. The reverse command to the SP locking is SPContUnlockBySP.

In the SP locked life cycle state, Root may decide to lock the SP Container too, by issuing the command SPContLockByRoot and changes the life cycle state of the SP Container to MC_CONT_STATE_ROOT_SP_LOCKED. In this state only Root is able to change the life cycle state of the SP Container back to MC_CONT_STATE_SP_LOCKED by issuing the command SPContUnlockByRoot.

In each valid SP Container life cycle state, Root may issue the command SPContUnregister to delete and free the SP Container and all dependent TA Containers of that Service Provider.

7.5.4 TA Container Life Cycle State Transitions

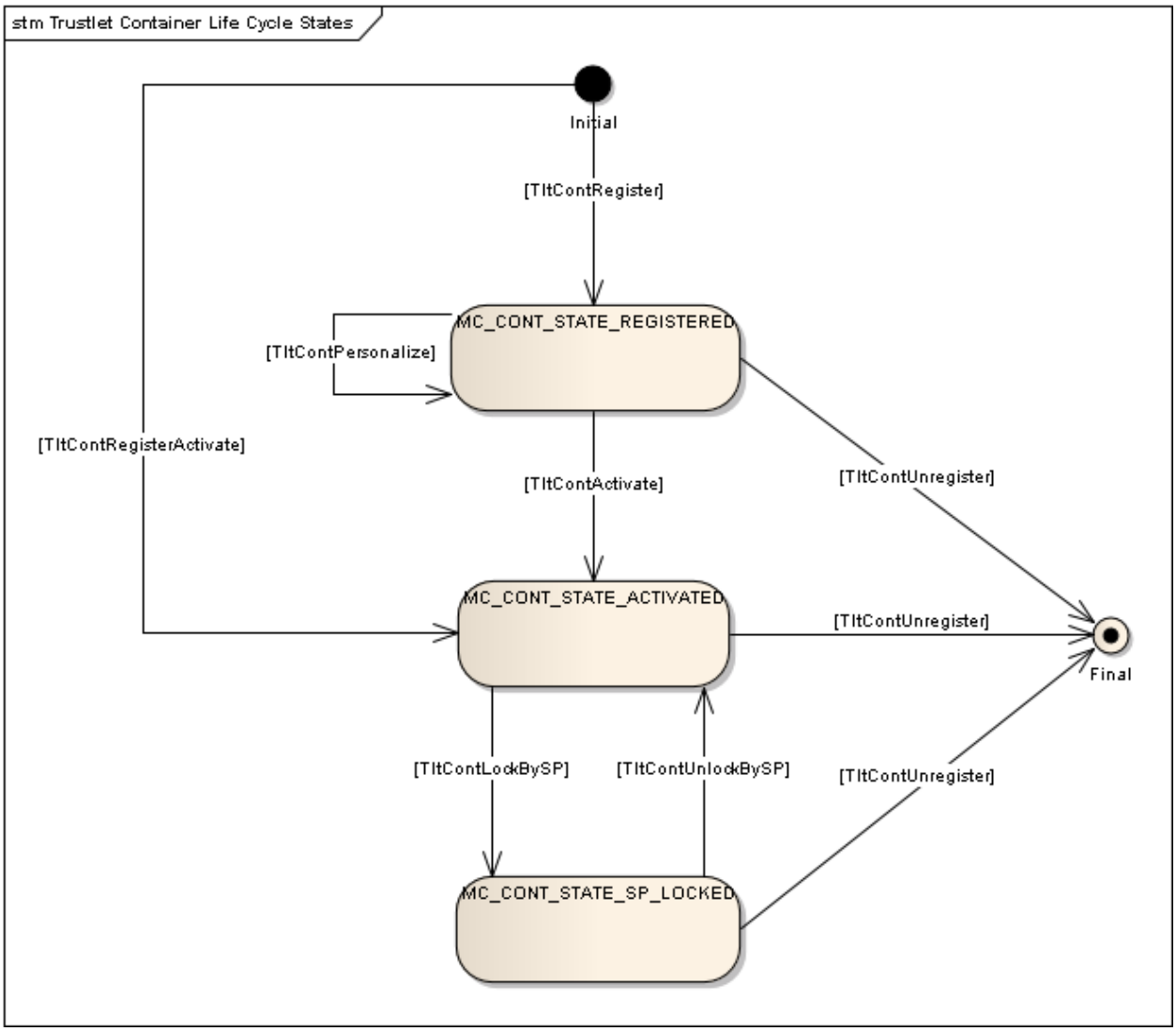


Figure 10: TA Container Life Cycle State Transitions.

A SP TA Container may be created by the Service Provider administrative command TltContRegister. The life cycle state of that TA Container is set to MC_CONT_STATE_REGISTERED. In this state the TA Container is not yet functional and may be personalized with additional personalization data by transmitting 0, 1 or more TltContPersonalize commands. When the personalization process is finished, SP changes the life cycle state of the TA Container to the fully functional state MC_CONT_STATE_ACTIVATED by issuing the command TltContActivate.

If no separate personalization phase for the TA is required, the TA Container may be directly created and activated by issuing the command TltContRegisterActivate.

Due to business or security reasons the SP may decide to explicitly lock its TA Container with TltContLockBySP (MC_CONT_STATE_SP_LOCKED). In this state the TA may not be started in SWd. The TA returns back to the activated state, when the SP issues TltContunlockBySP.

In each valid TA Container life cycle state SP may issue TltContUnregister, and delete and free the TA Container.

7.5.5 Implicit and Explicit Container Locks

As general rule Root administrative lock commands (RootContLockByRoot, SPContLockByRoot) do not have an influence on the functionality of the Root Container itself, but on all objects in the sub-tree below the Root Container (SP Container, TA Container). As well as SP administrative lock commands (SPContLockBySP, TltContLockBySP) do not have an influence on the functionality of the SP Container, but on the TA Container of that SP.

When Root or a Service Provider issues a ...Lock... command to the appropriate Container type (Root, SP or TA Container), that Container gets **explicitly** locked. That Container gets unlocked again by issuing the affiliated ...Unlock... command.

But there are also situations, that a Container gets locked, even when the Container is not explicitly locked. These locks are called **implicit** locks. E.g. when a SP Container gets locked (by Root or by SP), all dependent TA Containers of that SP Container, which are not explicitly locked, gets implicitly locked. The explicitly locked TA Containers of that SP retain their explicit locked state.

When that SP Container gets unlocked again, all implicitly locked TA Containers of that SP gets unlocked too. The explicitly locked TAs of that SP retain their explicit locked state.

A Root lock (RootContLockByRoot) cascades the implicit lock over all SP and TA Containers into the sub-hierarchy of the Root Container tree.

7.6 SECURE OBJECTS

All content objects are stored in the NWd registry as Secure Objects (see chapter *Secure Objects*). The following table shows the mapping between Content Object and Secure Objects as stored in the NWd:

Table 9: Content Objects and Secure Objects

Content Object	Secure Object Naming
Authentication Token	SO.AuthToken
Root Container	SO.RootCont
SP Container	SO.SPCont
TA Container	SO.TACont
Personalization Data Container	SO.PersoDataCont

The plaintext parts of the Content Objects are mapped to the *Plaintext Data* of the User Data in the Secure Object. The encrypted parts of Content Objects are mapped to the *Data to be encrypted* part of User Data of the Secure Object (see *Secure Object Format*).

Note:

The NWd can only read the plaintext parts of Secure Objects (like Type, Version, Link-Information ...). Modification operations on the Secure Objects by the NWd are discovered because of the integrity protection of the Secure Objects.

8 CONTENT MANAGEMENT OPERATIONS

8.1 DEVICE BINDING

DEVICE BINDING is an operation taking place in the production site of the OEM. This operation is a prerequisite for all other operations.

Participating roles are Root, TEE Vendor and OEM. A KPH of the TEE Vendor containing vendor specific private keys and role dependent public keys support the process at the OEM site.

The SoC Authentication key K.SoC.Auth is created by the KPH and bound to the device identifying SUID. The encrypted Secure Object of the Authentication Token is transferred to TEE Vendor and Root for backup purposes. SUID and K.Soc.Auth are transmitted to Root backend system via the TEE Vendor backend system.

Content Management Commands:

- GetSUID
- GenerateAuthToken

Precondition

<t-base can only run on a SoC which is providing:

- SUID
- K.Device.Fuse (SoC hardware key)

KPH is providing:

- PuK.Root.Transport to encrypt the SoC individual key K.SoC.Auth. This enciphered data can only be deciphered by Root with PrK.Root.Transport
- PuK.Vendor.MCAct to encipher the corresponding SUID and SO.AuthToken. Therefore the reference between SUID and K.SoC.Auth can only be identifiable for the recipient who is in possession of key Prk.Vendor.MCAct.
- PrK.Kph.Request bound to identifier KID to generate a signature of the GenerateAuthToken command.
- PrK.Vendor.Receipt to create signature which shall be created from KPH. TEE vendor has to verify this signature in the receipt for Authenticity

TEE vendor is in possession of:

- PuK.Vendor.Receipt to verify the signature in the receipt created by KPH for Authenticity

<t-directory (Root) is in possession of:

- PrK.Vendor.MCAct to decipher the SUID and SO.AuthToken
- PrK.Root.Transport to decipher the K.SoC.Auth

Overview

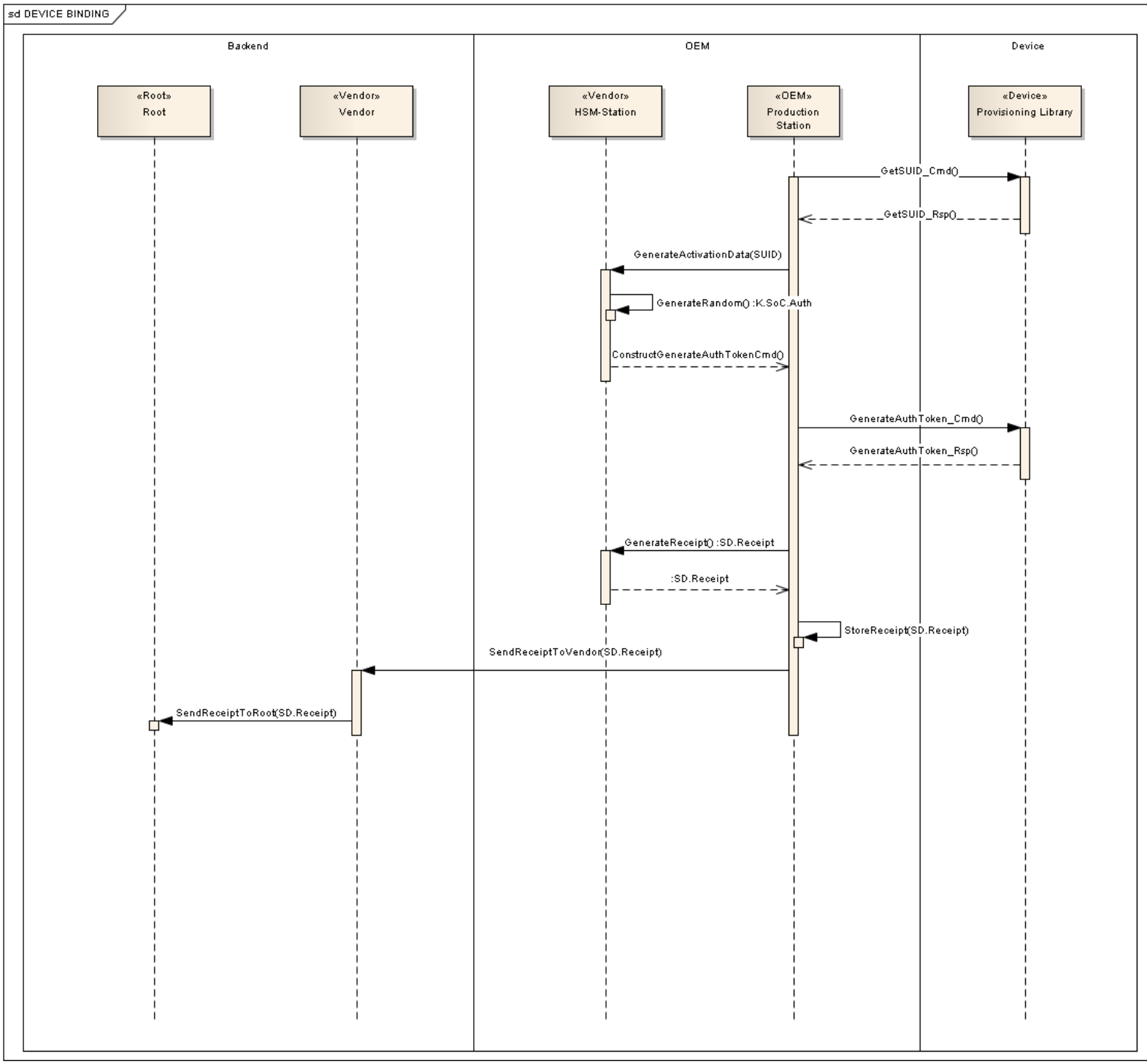


Fig. 1 DEVICE BINDING

The OEM Production Station requests from the <t-base Provisioning Library the SUID of the SoC (*GetSUID*).

The Production Station requests the KPH to generate Activation Data (GenerateActivationData) providing the SUID of the SoC. The KPH creates a random value, which is used as SoC Authentication Key K.SoC.Auth, and constructs the

GenerateAuthToken command (*GenerateAuthToken*) for the Provisioning Library. The command *GenerateAuthToken* is signed by KPH with key PrK.Kph.Request bound to identifier KID.

The Production Station transmits the GenerateAuthToken command to the Provisioning Library.

The Provisioning Library verifies the cryptographic security of the command and then creates the Secure Object of the Authentication Token (SO.AuthToken) and may store that Secure Object in the persistent storage of the device for later authentication. That SO is returned in the response to the Production Station. Alternatively that Secure Object could also be stored on the device by the Production Station of the OEM.

The Production Station transmits that SO.AuthToken to the KPH and requests the generation of a receipt (GenerateReceipt). The Receipt contains:

- K.SoC.Auth encrypted with a Root public key
- SUID and SO.AuthToken encrypted with a Vendor public key.
- Signature of the above data shall be generated with a TEE Vendor private key

The signature of the receipt may be verified by Vendor with the appropriate public key.

Only Root is able to decrypt K.SoC.Auth with its appropriate private key (Note: K.SoC.Auth in SO.AuthToken is only encrypted with a key derived from the device master key).

The Root is able to decrypt SO.AuthToken with the appropriate Vendor private key. The Secure Object of the Authentication Token is stored by Root for backup purpose. The NWd Secure Objects may be reinstalled by Root, when this data gets corrupt on the device.

Vendor transmits the Root encrypted K.SoC.Auth and the Vendor encrypted SUID and SO.AuthToken to Root. Root is now able to do a SOC AUTHENTICATION with SUID and K.SoC.Auth and a subsequent ROOT REGISTRATION.

8.2 ROOT OPERATIONS

The Root operations are performed between the <t-directory Root backend and the Content Management Trusted Application (CMTA).

<t-directory sends the Content Management Protocol commands to the CMTA through a proxy running in the NWd which is called the Root-PA. The Root-PA is also taking care of storing Secure Objects for the CMTA in a registry folder of the NWd.

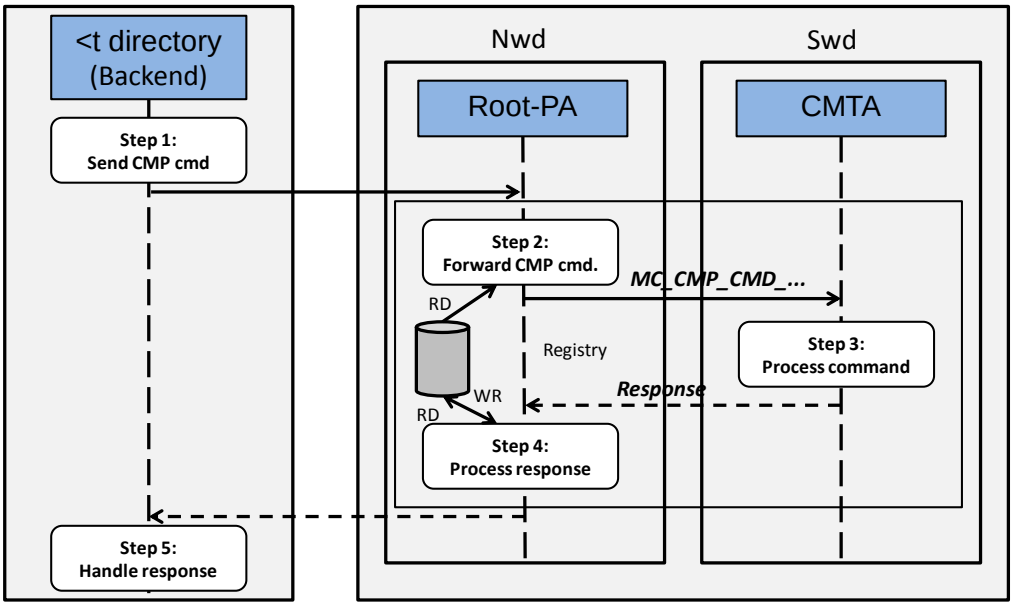


Figure 11: Root Operations Flow.

8.2.1 SOC AUTHENTICATION

Root must authenticate the SoC before performing the operation ROOT REGISTER ACTIVATE.

SOC AUTHENTICATION is a mutual authentication between Root and the SoC (the device) with the SoC Authentication Key K.SoC.Auth. After the authentication a secure channel is established between these two entities.

Participating entities are Root, the NWd Provisioning Agent (resp. its TA Connector), the SWd Content Management TA and <t-base.

Content Management Commands:

- BeginSoCAuthentication
- Authenticate

Precondition

The Secure Object of the Authentication Token including K.SoC.Auth had been established in a previous DEVICE BINDING operation and must be consistent stored as a Secure Object in NWd. After DEVICE BINDING, K.SoC.Auth is known only to Root and <t-base.

Overview

The Root backend prepares the *BeginSoCAuthentication* command and transmits the high level command to the Provisioning Agent.

In step 2 the PA extracts the Secure Object of the Authentication Token (SO.AuthToken) from the registry and sends it with the *BeginSoCAuthentication* command to the Content Management TA.

In step 3 CMTA in cooperation with <t-base checks the consistency of the Authentication Token and unwraps the Secure Object to get the SoC Authentication Key K.SoC.Auth. Two random values RND1 and K1 are generated. RND1 is used to detect replay attacks on the <t-base side. RND1 and K1 are used later in the key derivation process for the content management session keys SKi and SKc and the Secure Sequence Counter SSC. <t-base generates the command response, which contains mainly plaintext SUID and RND1 together with a MAC value of this data generated with K.SoC.Auth.

The PA returns the response to Root.

In step 5 Root checks the consistency of response, extracts SUID and finds in the backend system the affiliated K.SoC.Auth. K.SoC.Auth is used to check the MAC value of response.

In the second loop Root generates the *Authenticate* command. In step 1 two random values RND2 and K2 are generated. RND2 is used to detect replay attacks on Root side. RND2 and K2 are later part of the content management session key derivation.

PA forwards in step 2 Root's *Authenticate* command to the CMTA.

In step 3 CMTA checks the consistency of command and <t-base decrypts the message with K.SoC.Auth. It extracts RND1 from the message and compares it to the internally stored RND1 value. As RND1 is a random value, a replayed command message would provide a different RND1 value.

CMTA uses K.SoC.Auth, <t-base RND1 and K1, Root RND2 and K2 to derive the content management session keys SKi and SKc and the session counter. The security state on <t-base side is set to AUTH_SOC.

<t-base generates the response message, which contains <t-base RND1 and K1 and Root RND2, encrypted with K.SoC.Auth.

PA transmits the response to Root.

In step 5 Root checks the consistency of response, decrypts the response with K.SoC.Auth and extracts <t-base RND1 and K1 and Root RND2. Root compares the response RND2 with the internally stored RND2 and is able to detect command replays.

Root processes now <t-base RND1 and K1 and Root RND2 and K2 and derives with K.Soc.Auth the content management session keys Ski, SKc and the send sequence counter SSC.

After that, the secure channel between Root and CMTA is established.

8.2.2 ROOT AUTHENTICATION

ROOT AUTHENTICATION is a mutual authentication between Root and <t-base, which establishes a secure channel between these two entities for further content management operations (Root administrative operations).

Participating entities are Root, the NWd Provisioning Agent (PA, resp. it's TA Connector), the SWd Content Management TA (CMTA) and <t-base (MC).

Prerequisite for ROOT AUTHENTICATION is the existence of Root Container including the Root Authentication Key K.Root.Auth, which had been created once within the *ROOT REGISTER ACTIVATE* operation.

Content Management Commands:

- BeginRootAuthentication
- Authenticate

Precondition

Secure Object Root Container (SO.RootCont) must be present and consistent in the NWd.

Overview

The Root backend prepares the *BeginRootAuthentication* command and transmits the high level command to the Provisioning Agent.

In step 2 the PA extracts the Secure Object of the Root Container (SO.RootCont) from the registry, and sends it with the *BeginRootAuthentication* command to the Content Management TA.

In step 3 CMTA in cooperation with <t-base checks the consistency of the Root Container and unwraps the Secure Object to get the Root Authentication Key K.Root.Auth. Two random values RND1 and K1 are generated. RND1 is used to detect replay attacks on the <t-base side. RND1 and K1 are used later in the key derivation process for the content management session keys SKi and SKc and the Secure Sequence Counter SSC. <t-base generates the command response, which contains mainly plaintext SUID and RND1 together with a MAC value of this data generated with K.Root.Auth.

The PA returns the response to Root.

In step 5 Root checks the consistency of response, extracts SUID and finds in the backend system the affiliated K.Root.Auth. K.Root.Auth is used to check the MAC value of response.

In the second loop Root generates *Authenticate* command. In step 1 two random values RND2 and K2 are generated. RND2 is used to detect replay attacks on Root side. RND2 and K2 are later part of the content management session key derivation.

PA forwards in step 2 Root's *Authenticate* command to the CMTA.

In step 3 CMTA checks the consistency of command and <t-base decrypts the message with K.Root.Auth. It extracts RND1 from the message and compares it to the internally stored RND1 value. As RND1 is a random value, a replayed command message would provide a different RND1 value.

CMTA uses K.Root.Auth, <t-base RND1 and K1, Root RND2 and K2 to derive the content management session keys SKi and SKc and the session counter SSC. The security state on <t-base side is set to AUTH_ROOT.

<t-base generates the response message, which contains <t-base RND1 and K1 and Root RND2, encrypted with K.Root.Auth.

PA transmits the response to Root.

In step 5 Root checks the consistency of response, decrypts the response with K.Root.Auth and extracts <t-base RND1 and K1 and Root RND2. Root compares the response RND2 with the internally stored RND2 and is able to detect command replays.

Root processes now <t-base RND1 and K1 and Root RND2 and K2 and derives with K.Root.Auth the content management session keys SKi, SKc and the send sequence counter SSC.

After that, the secure channel between Root and CMTA is established.

8.2.3 ROOT REGISTER ACTIVATE

After SOC AUTHENTICATION the operation ROOT REGISTRATION ACTIVATE generates and activates the Root Container which includes the Root Authentication Key K.Root.Auth. After ROOT REGISTRATION ACTIVATION all data objects necessary for administration by Root are created. Root may now be authenticated by ROOT AUTHENTICATION; a further SOC AUTHENTICATION is not possible any more.

Participating entities are Root, the NWd Provisioning Agent (PA, resp. it's TA Connector), the SWd Content Management TA (CMTA) and <t-base (MC).

Content Management Commands

- RootContRegisterActivate

Precondition

- A previous DEVICE BINDING took place. The Root Container is not yet registered.
- Root processed a SOC AUTHENTICATION with the SoC Authentication Key, the Security State is AUTH_SOC.
- SOC AUTHENTICATION created a secure channel, which is cryptographically secured by the session keys:
- SKi (integrity protection)
- SKc (confidentiality)
- Send Sequence Counter (SSC)

Overview

After SOC AUTHENTICATION in step 1 Root generates a high level *RootContRegisterActivate* command and transmits the command to the Provisioning Agent PA. The command mainly contains the cryptographically secured Root Authentication Key K.Root.Auth.

In step 2 PA only transfers the command to the CMTA.

In step 3 CMTA in cooperation with <t-base checks the consistency of the command, checks that the current security state is AUTH_SOC and verifies the MAC of the message using SKi and SSC, created in a previous SOC AUTHENTICATION. K.Root.Auth is decrypted.

CMTA creates the Root Container including SUID and K.Root.Auth and sets the life cycle state of the Root Container to MC_CONT_STATE_ACTIVATED (see *Root Container Life Cycle State Transitions)*. <t-base wraps the Root Container with the System Context key into the Root Container Secure Object (SO.RootCont).

CMTA generates the command response, which mainly contains the Secure Object of the Root Container SO.RootCont cryptographically protected.

After creation of the Root Container, the Authentication Token shall not be used any more.

In step 4 PA stores the Secure Object of the Root Container in <t-base registry and removes the Authentication Token to prevent subsequent SOC AUTHENTICATION. PA transfers the response to Root.

In step 5 Root checks the consistency of command response, checks the security using SKi and SSC and decrypts response to get the Secure Object of the Root Container.

8.2.4 ROOT UNREGISTRATION

The ROOT UNREGISTRATION operation is the reverse operation of *ROOT REGISTER ACTIVATE*.

After ROOT UNREGISTRATION all TA personalization data, TA and TA Container, Service Provider Container and Root Container are deleted and the status of <t-base is equivalent to the status after Device Binding.

The Security State shall be set to AUTH_NONE finally.

Used Content Management Commands

- RootContUnregister

Precondition

- The Root Container had already been installed.
- ROOT AUTHENTICATION had been processed with Root Authentication Key K.Root.Auth. The security status is AUTH_ROOT.
- ROOT AUTHENTICATION created a secure channel, which is cryptographically secured by the session keys:
- SKi (integrity protection)
- SKc (confidentiality)
- Send Sequence Counter (SSC)

Overview

A previous ROOT AUTHENTICATION created a secure channel between Root backend and the CMTA.

In step 1 Root generates the high level command message, which mainly contains the SUID of the device in combination with a MAC of the command. Root backend transmits the message to the Provisioning Agent PA. In step 2 PA transmits this command to the CMTA.

In step 3 the CMTA checks the consistency of the message and verifies that a previous ROOT AUTHENTICATION took place, which set the security state to AUTH_ROOT. <t-base checks the MAC value of the command. CMTA compares the provided SUID of the command against the SUID gained from <t-base. CM Applet creates the command response, which mainly contains the SUID and a MAC.

In step 4 PA deletes all Secure Objects and SP TAs. The response is transmitted to the Root backend.

Root may remove all stored Secure Objects (except SO.AuthToken) belonging to that device with SUID, as they are not valid any more.

After ROOT UNREGISTRATION no ROOT AUTHENTICATION is possible any more.

8.2.5 ROOT LOCK

The Root administrative operation ROOT LOCK sets the life cycle state of the Root Container to MC_CONT_STATE_ROOT_LOCKED (see *Root Container Life Cycle State Transitions*).

When the Root Container is locked, all dependent SP Container and TA Container in the sub-hierarchy of the Root Container are implicitly locked too (on explicit and implicit container locks see chapter *Implicit and Explicit Container Locks*). That means after a Root lock no further SP administrative commands may be issued.

As a result only ROOT AUTHENTICATION and ROOT UNLOCK are further possible as administrative operations. No Service Provider TAs are executable in that life cycle state.

Only the authenticated Root is able to unlock the Root Container by the operation *ROOT UNLOCK*.

Content Management Commands

- RootContLockbyRoot

8.2.6 ROOT UNLOCK

The Root administrative operation ROOT UNLOCK resets the life cycle state of the Root Container to MC_CONT_STATE_ACTIVATED (see *Root Container Life Cycle State Transitions*).

When the Root Container is unlocked, all dependent SP Container and TA Container in the sub-hierarchy of the Root Container are implicitly unlocked too (on explicit and implicit container locks see chapter *Implicit and Explicit Container Locks*). Further Root and SP Administrative operations are possible again.

Only the authenticated Root is able to unlock the Root Container by the operation *ROOT UNLOCK*.

Content Management Commands

- RootContUnlockByRoot

8.2.7 SP REGISTRATION

The authenticated Root creates in the operation SP REGISTRATION a SP Container of a Service Provider SP_ID with its SP Authentication Key K.SP.Auth. As the Root Container links to the SP Container via SP_ID, Root Container needs to be adapted accordingly in the SP

REGISTRATION operation. Finally a Secure Object of the SP Container SO.SPCont and an adapted Root Container SO.RootCont is stored in <t-base registry. After SP REGISTRATION and SP ACTIVATION SP is able to authenticate to <t-base (SP AUTHENTICATION) and to install SP TAs.

Content Management Commands

- SPContRegister

Precondition

- Security State must be AUTH_ROOT
- Root Container must be activated

Overview

Root backend starts the SP REGISTRATION operation by creating a secure channel with session keys SKi and SKc and the sequence counter SSC.

In step 1 Root backend generates the SP registration message. The SP registration message mainly contains the encrypted SP Authentication Key K.SP.Auth and the Service Provider ID SP_ID. The Root backend transmits the message to the Provisioning Agent PA.

In step 2 PA sets up the *SPContRegister* command with SP_ID, the encrypted K.SP.Auth.

In step 3 CMTA checks that a previous ROOT AUTHENTICATION took place and unwraps the command. CMTA creates the SP Container containing SP_ID, K.SP.Auth, sets the life cycle state to MC_CONT_STATE_REGISTERED (see *SP Container Life Cycle State Transitions*) and let <t-base wrap that container into the Secure Object SO.SPCont. The link to the SP Container (SP_ID) is updated in the Root Container and the Root Container wrapped into the Root Container Secure Object SO.RootCont.

CMTA creates the command response, which mainly contains the encrypted Secure Objects of the Root Container and the SP Container and the SKi MACed message.

In step 4 the PA stores in <t-base registry the Secure Objects of Root and the SP Container and transmits the response back to the Root backend system.

In step 5 Root checks the consistency of the response.

8.2.8 SP UNREGISTRATION

SP UNREGISTRATION is the reverse operation to *SP REGISTRATION*.

The operation SP UNREGISTRATION removes the SP Container of a SP identified by SP_ID and all meanwhile installed corresponding TA Containers, TAs and TA Personalization Data. The link to the Service Provider Container identified by SP_ID is deleted from the Root Container.

For SP UNREGISTRATION a previous Root Authentication step is necessary.

Content Management Commands

- SPContUnregister

Precondition

- Security State must be AUTH_ROOT
- Root Container must be activated

Overview

Root successfully initiated a secure channel with ROOT AUTHENTICATION.

In step 1 Root generates an SP Unregistration message which mainly contains the SP_ID of the SP to be unregistered.

In step 2 the Provisioning Agent PA creates the command with SP_ID and transmits the command to the CMTA in SWd.

In step 3 CMTA checks whether a previous ROOT AUTHENTICATION took place and the security state is set to AUTH_ROOT. <t-base checks the cryptographic security of the command and unlinks the SP Container from the Root Container. The Root Container is then wrapped into its Secure Object SO.RootCont. The SP_ID together with the updated SO.RootCont is basis for the construction of the command response.

In step 4 the PA removes in <t-base registry all Secure Objects of the dependent SP TA Containers, TA codes, Personalization Data and the SP Container of the Service Provider SP_ID. The updated Secure Object of the Root Container is stored in the registry. PA transmits the command response back to the ROOT backend.

In step 5 Root checks the cryptographic validity of the command response.

8.2.9 SP REGISTER ACTIVATE

The SP REGISTER ACTIVATE operation is a combined *SP REGISTRATION* and *SP AUTHENTICATION*

SP AUTHENTICATION is a mutual authentication between Service Provider and <t-base, which establishes a secure channel between these two entities for further Service Provider administrative use.

Participating entities are the Service Provider, the NWd Provisioning Agent (PA, resp. it's TA Connector), the SWd Content Management TA (CMTA) and <t-base (MC).

Prerequisite for SP AUTHENTICATION is the existence of activated Root Container and activated Service Provider Container containing the SP Authentication Key K.SP.Auth. The SP Container had been created in a previous Service Provider registration step.

Content Management Commands

- BeginSPAuthentication
- Authenticate

Precondition

- Secure Object SP Container (SO.SPCont) must be present and consistent in the NWd

- Root Container must be activated

Overview

The SP backend prepares *BeginSPAuthentication* command (see step 1 of the) and transmits the high level command to the Provisioning Agent.

In step 2 the PA extracts the Secure Object of the SP Container (SO.SPCont) and sends it with the *BeginSPAuthentication* command to the Content Management TA.

In step 3 CMTA in cooperation with <t-base checks the consistency of the SP Container and unwraps the Secure Object to get the SP Authentication Key K.SP.Auth. Two random values RND1 and K1 are generated. RND1 is used to detect replay attacks on the <t-base side. RND1 and K1 are used later in the key derivation process for the content management session keys SKi and SKc and the Secure Sequence Counter SSC. <t-base generates the command response, which contains mainly plaintext SUID and RND1 together with a MAC value of this data generated with K.SP.Auth.

The PA returns the response to the SP.

In step 5 the SP checks the consistency of the response, extracts SUID and finds in the backend system the affiliated K.SP.Auth. K.SP.Auth is used to check the MAC value of response.

In the second loop SP generates *Authenticate* command. In step 1 two random values RND2 and K2 are generated. RND2 is used to detect replay attacks on SP side. RND2 and K2 are later part of the content management session key derivation.

PA forwards in step 2 SP's *Authenticate* command to the CMTA.

In step 3 CMTA checks the consistency of command and <t-base decrypts the message with K.SP.Auth. It extracts RND1 from the message and compares it to the internally stored RND1 value. As RND1 is a random value, a replayed command message would provide a different RND1 value.

CMTA uses K.SP.Auth, <t-base RND1 and K1, Root RND2 and K2 to derive the content management session keys SKi and SKc and the session counter SSC. The security state on <t-base side is set to AUTH_SP.

<t-base generates the response message, which contains <t-base RND1 and K1 and SP RND2, encrypted with K.SP.Auth.

PA transmits the response to SP.

In step 5 SP checks the consistency of response, decrypts the response with K.SP.Auth and extracts <t-base RND1 and K1 and SP RND2. SP compares the response RND2 with the internally stored RND2 and is able to detect command replays.

SP processes now <t-base RND1 and K1 and SP RND2 and K2 and derives with K.SP.Auth the content management session keys Ski, SKc and the send sequence counter SSC.

After that, the secure channel between the SP and CMTA is established.

SP ACTIVATION operation. It directly creates the SP Container and updates it life cycle state to MC_CONT_STATE_ACTIVATED (see *SP Container Life Cycle State Transitions*). After this operation the SP Container is fully functional.

Content Management Commands

- SPContRegisterActivate

Precondition

- Security State must be AUTH_ROOT
- Root Container must be activated

8.2.10 SP LOCK BY ROOT

Due to business or security reasons Root may decide to lock the SP Container of a Service Provider. The Root administrative operation SP LOCK BY ROOT locks the Service Provider Container. The new life cycle state is either MC_CONT_STATE_ROOT_LOCKED or MC_CONT_STATE_ROOT_SP_LOCKED depending on the state it had before the Root lock (see also *SP Container Life Cycle State Transitions*). As result the Service Provider is not able process any Service Provider administrative operations. All dependent TAs of the Service Provider are not executable anymore in the new life cycle state. Only the authenticated Root is able to unlock the Service Provider Container by the operation *SP UNLOCK BY ROOT*.

Content Management Commands

- SPContLockByRoot

Precondition

- Security State must be AUTH_ROOT

8.2.11 SP UNLOCK BY ROOT

The operation SP UNLOCK BY ROOT unlocks the previously locked Service Provider Container by Root. The life cycle state changes back to the state it had before the Root lock, i.e. MC_CONT_STATE_ACTIVATED or MC_CONT_STATE_SP_LOCKED (see also *SP Container Life Cycle State Transitions*). Only the authenticated Root is able to unlock the Service Provider Container.

When the life cycle state changes back to MC_CONT_STATE_ACTIVATED all implicitly locked SP TAs (see also *Implicit and Explicit Container Locks*) are executable again. All SP administrative operations may be processed again by the Service Provider.

Content Management Commands

- SPContunlockByRoot

Precondition

- Security State must be AUTH_ROOT

8.3 SERVICE PROVIDER OPERATIONS

The Service Provider operations are performed between the SP backend (or TSM) and the Content Management Trusted Application (CMTA).

The Service Provider sends the Content Management Protocol commands to the CMTA through a proxy running in the NWd which is called the Provisioning Agent (PA). The PA is also taking care of storing Secure Objects in a registry folder of the NWd.

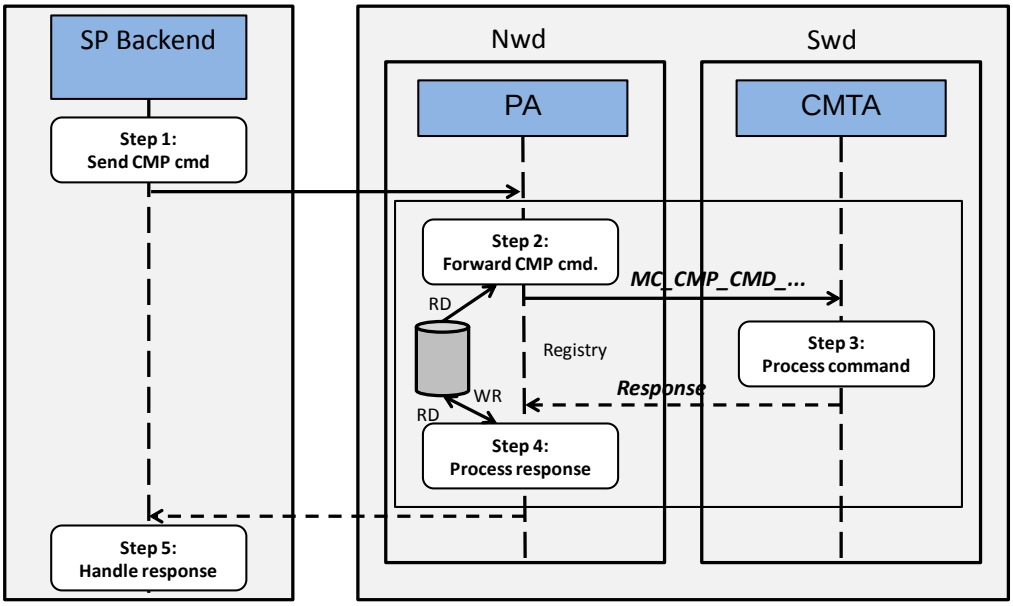


Figure 12: SP Operations Flow.

8.3.1 SP AUTHENTICATION

SP AUTHENTICATION is a mutual authentication between Service Provider and <t-base, which establishes a secure channel between these two entities for further Service Provider administrative use.

Participating entities are the Service Provider, the NWd Provisioning Agent (PA, resp. it's TA Connector), the SWd Content Management TA (CMTA) and <t-base (MC).

Prerequisite for SP AUTHENTICATION is the existence of activated Root Container and activated Service Provider Container containing the SP Authentication Key K.SP.Auth. The SP Container had been created in a previous Service Provider registration step.

Content Management Commands

- BeginSPAuthentication
- Authenticate

Precondition

- Secure Object SP Container (SO.SPCont) must be present and consistent in the NWd
- Root Container must be activated

Overview

The SP backend prepares *BeginSPAuthentication* command (see step 1 of the) and transmits the high level command to the Provisioning Agent.

In step 2 the PA extracts the Secure Object of the SP Container (SO.SPCont) and sends it with the *BeginSPAuthentication* command to the Content Management TA.

In step 3 CMTA in cooperation with <t-base checks the consistency of the SP Container and unwraps the Secure Object to get the SP Authentication Key K.SP.Auth. Two random values RND1 and K1 are generated. RND1 is used to detect replay attacks on the <t-base side. RND1 and K1 are used later in the key derivation process for the content management session keys SKi and SKc and the Secure Sequence Counter SSC. <t-base generates the command response, which contains mainly plaintext SUID and RND1 together with a MAC value of this data generated with K.SP.Auth.

The PA returns the response to the SP.

In step 5 the SP checks the consistency of the response, extracts SUID and finds in the backend system the affiliated K.SP.Auth. K.SP.Auth is used to check the MAC value of response.

In the second loop SP generates *Authenticate* command. In step 1 two random values RND2 and K2 are generated. RND2 is used to detect replay attacks on SP side. RND2 and K2 are later part of the content management session key derivation.

PA forwards in step 2 SP's *Authenticate* command to the CMTA.

In step 3 CMTA checks the consistency of command and <t-base decrypts the message with K.SP.Auth. It extracts RND1 from the message and compares it to the internally stored RND1 value. As RND1 is a random value, a replayed command message would provide a different RND1 value.

CMTA uses K.SP.Auth, <t-base RND1 and K1, Root RND2 and K2 to derive the content management session keys SKi and SKc and the session counter SSC. The security state on <t-base side is set to AUTH_SP.

<t-base generates the response message, which contains <t-base RND1 and K1 and SP RND2, encrypted with K.SP.Auth.

PA transmits the response to SP.

In step 5 SP checks the consistency of response, decrypts the response with K.SP.Auth and extracts <t-base RND1 and K1 and SP RND2. SP compares the response RND2 with the internally stored RND2 and is able to detect command replays.

SP processes now <t-base RND1 and K1 and SP RND2 and K2 and derives with K.SP.Auth the content management session keys Ski, SKc and the send sequence counter SSC.

After that, the secure channel between the SP and CMTA is established.

8.3.2 SP ACTIVATION

In the SP administrative operation SP ACTIVATION the Service Provider activates its SP Container and replaces the authentication key K.SP.Auth by its own authentication key. The SP Activation operation changes the life cycle state of the SP Container from MC_CONT_STATE_REGISTERED to MC_CONT_STATE_ACTIVATED (see *SP Container Life Cycle State Transitions*).

After the activation step all other administrative operations of the Service Provider are then available.

Content Management Commands

- SPContActivate

Precondition

- Security State must be AUTH_SP
- Root Container must be activated

8.3.3 SP LOCK BY SP

Due to business or security reasons an authenticated Service Provider may decide to lock its SP Container.

The Service Provider administrative operation SP LOCK BY SP changes the life cycle state of the SP Container from MC_CONT_STATE_ACTIVATED to MC_CONT_STATE_SP_LOCKED (see *SP Container Life Cycle State Transitions*). As result all dependent TAs of that Service Provider are not executable anymore. Only the authenticated Service Provider is able to unlock the Service Provider Container by the operation *SP UNLOCK BY SP*.

Content Management Commands

- SPContLockBySP

Precondition

- Security State must be AUTH_SP

8.3.4 SP UNLOCK BY SP

The operation SP UNLOCK BY SP unlocks the SP locked Service Provider Container and changes its life cycle state back to MC_CONT_STATE_ACTIVATED (see *SP Container Life Cycle State Transitions*).

All implicitly locked TA Containers (see *Implicit and Explicit Container Locks*) gets unlocked too. The explicit locked TA Containers retain their explicit locked state. All SP administrative operations may be processed again.

Content Management Commands

- SPContUnlockBySP

Precondition

- Security State must be AUTH_SP

8.3.5 TA REGISTRATION

After SP AUTHENTICATION of the Service Provider with ID SP_ID the operation TA REGISTRATION generates a TA Container for a TA of the Service Provider. The newly created TA Container contains the Service Provider SP_ID, the UUID of the TA and a symmetric Service Provider key K.SP.TltEnc, which is used to decrypt the code of the TA. The life cycle state of that TA Container is set to MC_CONT_STATE_REGISTERED (see *TA Container Life Cycle State Transitions*). The key for signature verification of the TA is part of the affiliated SP Container.

During this operation a link from the SP Container to the newly created TA Container via UUID has to be established.

Content Management Commands

- TltContRegister

Precondition

- Security State must be AUTH_SP

Overview

The Service Provider backend system authenticates itself with SP AUTHENTICATION and a secure channel is established.

In step 1 the Service Provider creates a TA registration message, which contains the Service Provider ID SP_ID, the UUID of the new TA and the encrypted symmetric key K.SP.TltEnc, which is used to decrypt the TA code. The message is transmitted to the Provisioning Agent PA on the device.

In step 2 the PA creates the command containing SPID, UUID and the encrypted K.SP.TltEnc and transmits the command to the SWd CMTA.

In step 3 CMTA checks whether a previous SP AUTHENTICATION took place, the security state is set to AUTH_SP and checks the cryptographic security of the command. In cooperation with <t-base the encrypted key K.SP.TltEnc is decrypted and the Secure Object of the SP Container unwrapped.

CMTA generates the new TA Container containing SP_ID, UUID and K.SP.TltEnc and sets the life cycle state to MC_CONT_STATE_REGISTERED (see *TA Container Life Cycle State Transitions*). <t-base wraps that TA Container into its Secure Object SO.TACont. The UUID of the TA is registered in the SP Container and the updated SP Container wrapped into its Secure Object SO.SPCont. Finally the command response is created, which contains the Secure Objects of the SP Container and the newly created TA Container.

In step 4 PA stores the Secure Objects of SP and TA Container in <t-base registry. The response is transmitted to the SP backend.

In step 5 SP checks the cryptographic validity of the response.

8.3.6 TA UNREGISTRATION

The TA UNREGISTRATION operation is the reverse operation of *TA REGISTRATION* operation. After SP AUTHENTICATION, which creates a secure channel between the Service Provider backend and the CMTA, the operation unregisters (deletes) the TA Container of TA UUID and all affiliated Personalization Data associated to that TA Container. The link from the SP Container to the TA Container (UUID) is deleted.

Content Management Commands

- TltContUnregister

Precondition

- Security State must be AUTH_SP

Overview

After SP AUTHENTICATION in step 1 the Service Provider creates a TA unregistration message, which includes the Service Provider SPID and the UUID of the TA Container to be deleted. SP backend transmits the message to the Provisioning Agent PA on the device.

In step 2 the PA construct *TltContUnregister* command containing SPID and UUID and transmits it to the CMTA.

In step 3 the CMTA checks whether a previous SP AUTHENTICATION took place and checks the cryptographic security of the command. The UUID of the TA to be deleted is unregistered in the SP Container. Then the updated SP Container is wrapped in its Secure Object SO.SPCont. The CMTA creates the command response, which contains the SP_ID, UUID and the updated SO.SPCont.

In step 4 the PA stores the Secure Object of the SP Container in <t-base registry. The Secure Objects of the TA Container and all personalization data are deleted in the registry. Then the response is transmitted to the SP backend.

In step 5 SP checks the cryptographic validity of the response.

8.3.7 TA ACTIVATION

Before TA Activation the TA Container may be personalized with 0, 1 or more personalization data packets (see *TA PERSONALIZE*). In the TA ACTIVATION operation an already registered (and personalized) TA Container (see *TA REGISTRATION*) gets activated and changes its life cycle state to the fully functional state MC_CONT_STATE_ACTIVATED (see *TA Container Life Cycle State Transitions*).

Content Management Commands

- TltContActivate

Precondition

- Security State must be AUTH_SP

8.3.8 TA REGISTER ACTIVATE

When no TA Container personalization is required (see *TA PERSONALIZE*) the Service Provider may directly register and activate the respective TA Container. The TA REGISTER ACTIVATE operation is a combined TA REGISTRATION and TA ACTIVATION operation. It directly brings the TA Container into the life cycle state MC_CONT_STATE_ACTIVATED (see *TA Container Life Cycle State Transitions*).

Content Management Commands

- TltContRegisterActivate

Precondition

- Security State must be AUTH_SP

8.3.9 TA LOCK

Due to business or security reasons the authenticated Service Provider may decide to lock one of its TA Containers. The life cycle state of that TA Container is changed to MC_CONT_STATE_SP_LOCKED (see *TA Container Life Cycle State Transitions*). As result that TA of the Service Provider is not executable anymore. Only the authenticated Service Provider is able to unlock the TA Container by operation *TA UNLOCK*.

Content Management Commands

- TltContLockbySP

Precondition

- Security State must be AUTH_SP

8.3.10 TA UNLOCK

The Service Provider administrative operation TA UNLOCK unlocks a previously locked Service Provider TA Container. This operation is the reverse operation to TA LOCK. Only the authenticated Service Provider is able to change the life cycle state back to MC_CONT_STATE_ACTIVATED (see *TA Container Life Cycle State Transitions*). As result that TA of the Service Provider is executable again.

Content Management Commands

- TltContUnlockbySP

Precondition

- Security State must be AUTH_SP

8.3.11 TA PERSONALIZE

The operation PERSONALIZE generates a Secure Object of Personalization Data (SO.PersoDataCont) associated to a TA, which contains TA-specific personalization data. After Service Provider authentication and initiation of a secure channel, TA personalization data is transmitted within that secure channel to the CMTA. <t-base creates a Secure Object of that TA personalization data. This Secure Object is persisted by the Provisioning Agent PA in NWd <t-base registry.

Content Management Commands

- TltContPersonalize

Precondition

- Security State must be AUTH_SP

Overview

In step 1 after SP AUTHENTICATION, SP backend creates a personalization message with the appropriate TA personalization data. A personalization data identifier PID is added to the message, to identify that personalization data. The transmission of the personalization data to the Provisioning Agent PA is secured by the secure channel mechanism.

In step 2 the PA creates *TltContPersonalize* command and transmits the command to CMTA.

In step 3 the CMTA checks that the security state is AUTH_SP (a previous SP AUTHENTICATION occurred) and the cryptographic security of the command. The command is unwrapped.

The personalization data container is created; the data is identified with PID. <t-base wraps that personalization data in a Secure Object SO.PersoDataCont. PA creates the command response containing the secure Object of the personalization data.

In step 4 the Provisioning Agent stores the Secure Object SO.PersoDataCont in <t-base registry and transfers the response back to SP backend.

In step 5 SP checks the cryptographic validity of the response.